jc531 U.S. PTO
11/17/00

THE COMMISSIONER OF PATENTS AND TRADEMARKS
WASHINGTON, D.C 20231

Case Docket No.: 2225/1

Sir:

Transmitted herewith for filing is the patent application of

Inventor: Arye Barkan

For: A system and method for analyzing and coordinating Service-Level-Agreements (SLA) for Application-Service-Providers (ASP).

jc860 U.S. PTO
09/714204
11/17/00

Enclosed are:

- [x] 5 sheets of formal drawing(s).
- [x] An assignment of the invention to Oblicore Ltd
- [ ] A certified copy of a ________ application.
- [x] An associate power of attorney.
- [x] A verified statement to establish small entity status under 37 CFR 1 9 and 37 CFR 1 27.
- [ ] Other - _________

The filing fee has been calculated as shown below:

FOR:	(Col.1) NO.FILED	(Col 2) NO. EXTRA	SMALL ENTITY RATE	SMALL ENTITY FEE	OR	OTHER THAN A SMALL ENTITY RATE	OTHER THAN A SMALL ENTITY FEE
BASIC FEE				$ 355	OR		$ 710
TOTAL CLAIMS	17 - 20=		x9=	$	OR	x18=	$
INDEP CLAIMS	4	1	x40	$ 40	OR	x80	$
... Recordal of Assignment		40		$ 40	OR		$
			TOTAL	$ 435	OR	TOTAL	$

* If the difference in Col.1 is less than zero, enter "0" in Col 2

- [x] Please charge my Deposit Account No. 06-2140 in the amount of **$ 435**. A duplicate copy of this sheet is enclosed.
- [ ] A check in the amount of $___ to cover the filing fee is enclosed.
- [x] The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 06-2140.A duplicate copy of this sheet is enclosed.
  - [x] Any additional filing fees required under 37 CFR 1.16.
  - [x] Any patent application processing fees under 37 CFR 1.17.
- [x] The Commissioner is hereby authorized to charge payment of the following fees during the pendency of this application or credit any overpayment to Deposit Account No. 06-2140. A duplicate copy of this sheet is enclosed.
  - [x] Any patent application processing fees under 37 CFR 1.17.
  - [ ] The issue fee set in 37 CFR 1.18 at or before mailing of the Notice of allowance, pursuant to 37 CFR 1.311(b).
  - [x] Any filing fees under 37 CFR 1.16 for presentation of extra claims.

Respectfully,

Mark M. Friedman
Reg. No. 33,883

Mark M. Friedman
DR. MARK FRIEDMAN LTD.
C/O Anthony Castorina
2001 Jefferson Davis Highway
Suite 207
Arlington, Virginia 22202

SMALL BUSINESS CONCERN - NEW APPLICATION

Attorney Docket No.: 2225/1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In RE Application of: Arye Barkan

Filed Concurrently Herewith

For: A system and method for analyzing and coordinating Service-Level-Agreements (SLA) for Application-Service-Providers (ASP).

VERIFIED STATEMENT UNDER 37 CFR 1.27
CLAIMING STATUS AS A SMALL ENTITY

To The Commissioner of Patents and Trademarks:

I hereby declare that:

I am the owner of, or an official empowered to act on behalf of, the small business concern identified below:

Name of Concern: Oblicore Ltd.

Address : 7 Abba Hillel St. , Ramat Gan, Israel 52522

The small business concern identified above, together with its affiliates, employs fewer than 500 persons and qualifies as a small business concern as defined in 37 CFR 1.9(d) for purposes of paying reduced fees under 35 USC § 41(a) and § 41(b) to the Patent and Trademark Office with regard to the above-entitled invention described in the specification filed herewith.

Rights under contract or law have been conveyed to and remain with the small business concern identified above with regard to the above entitled invention.

If the rights held by the small business concern are not exclusive, each other party having rights to the invention is listed below, and no rights to the invention are held by any party who could not qualify as a small entity under 37 CFR 1.9(f), namely any person who could not be classified as an independent inventor under 37 CFR 1.9(c) if that person had made the invention, or any concern which would not qualify as a small business concern under 37 CFR 1.9(d) or a nonprofit organization under 37 CFR 1.9(e).

Full Name (Party 1) : NONE

Address : ______

Status : ☐ Individual ☐ Small Business Concern ☐ Nonprofit Organization

Full Name (Party 2) : ______

Address : ______

Status : ☐ Individual ☐ Small Business Concern ☐ Nonprofit Organization

I acknowledge the duty under 37 CFR 1.28(b) to file, in this application, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the issue fee due after the date on which status as a small entity is no longer appropriate.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application and any patent issuing thereon.

Amnon Madar		11 - November - 2000
Name of Person Signing	Signature	Date

Capacity of Person Signing: Controller

Address of Person Signing : 4 Zeelon Street, Rosh-Haain, Israel

APPLICATION FOR PATENT

Inventors: Arye Barkan, Aviv Freidin, , Lior Musman, Noam Rotem, Gal Baram, Dror Treves, Sharon Wagner

Title: A system and method for analyzing and coordinating Service-Level-Agreements (SLA) for Application-Service-Providers (ASP).

FIELD AND BACKGROUND OF THE INVENTION

The present invention relates to a system for enabling Service providers (including NSP – Network Service Providers, ISP – Internet Service Providers, etc. and primarily ASP – Application Service Providers), hereinafter referred to ASP's, to manage Service Level Agreements (hereinafter referred to as SLA's) in a highly efficient way. The purpose of the present invention is to give ASP's a tool to define SLA's with their customers, and to enable them to monitor and execute the actual service level given to a customer according to the SLA. The way this is achieved is through the application of a language based on Service Level Agreement Language of Measurement (hereinafter referred to as SLALOM).

An application service provider (ASP) is a company that uses the Internet to offer individuals or enterprises access to application programs and/or related services, so that these programs and services do not have to be located in personal or enterprise computers. The ASP market is becoming increasingly competitive. In order to survive, ASP's are beginning to offer a wider set of services to complement their offering. Besides the application that the ASP's are providing, they are offering Help Desk, technical support, consulting, training etc.

As this market matures, customers are demanding service level agreements (SLA) as the core component of the engagement, and reports on the actual service level delivered. Different customers will have different service level needs. The ASP's will be facing the problem of providing

differentiated services to different customers. Meeting the various service level expectations of its customers and allocating the sufficient amount of resources for each customer becomes increasingly complicated.

The ASP's that will survive this cutthroat competition will be those who:

- Offer tailored services to their customers
- Gain customers confidence in their service delivery
- Maintain an efficient operation

The Service Level Agreements (SLA) is a contract between a network service provider, such as an ASP, and a customer that specifies what and how services will be furnished by the network service provider. Common SLA's include the following factors: What percentage of the time services will be available, the number of users that can be served simultaneously, specific performance benchmarks to which actual performance will be periodically compared, the schedule for notification in advance of network changes that may affect users, help desk response time for various classes of problems, dial-in access availability, and usage statistics that will be provided[1]. The problem, however, is that these factors are difficult to measure and control. A new generation of tools has had to be created to deal with these developments.

There are tools today that compute service-levels, but each of those tools defines its own idea of service-level. There are many tools that measure different measurements that are used to compute the service-level. Some of these tools can compute certain aspects of service-level, but they lack the ability to define different types of service-level computation.

There is, however, no known tool in the market today that manages service level agreements (SLA's) for Asp's. There is also no tool that measures service-level, based on definitions in the SLA.

Tools that exist today enable raw measurements of resources or equipment that the ASP uses, but none of them allows a language that can describe the way to combine those raw measurements into a valuable description of the service level of a specific customer. Today, service providers that want to compute service-level that was given to a particular customer, must perform those calculations manually, using the raw measurements received from the measurement tools. This leads to very simple definitions of service-level, since more complicated definitions are very hard to compute manually.

Example of two such tools are InfoVista (http://www.infovista.com) and NetCool (http://www.netcool.com). Both these tools, gather information about resources on a network. Netcool focuses mainly on getting failure events from the network, in order to alarm the system administrator, while Infovista's main focus is on a longer term gathering of information, in order to produce reports of an overall performance.

There is thus a widely recognized need for, and it would be highly advantageous to have a tool that allows ASP's to define different ways to automatically compute service-levels, and consequently to measure and control SLA's.

The present invention allows such flexibility and automation. The invention, referred to hereinafter as Oblicore, is a central management tool, based on a specialized Service Level Agreement Language of Measurement (SLALOM) that allows the ASP to manage all aspects of the SLA's signed with its customers and to track the actual service level delivered. Using the present invention's solution, the ASP can provide timely and reliable reports to its customers on the service level delivered, compared with the service level agreed upon in the SLA. Another feature provided by the present invention is the calculation of penalties to be credited to the customer in case the

[1] www.gurunet.com, under Service Level Agreement

targets have not been met. The ASP that utilizes the present invention is able to optimize allocation of its resources according to customer prioritization and based on the actual service level delivered to each customer.

When the ASP industry becomes mainstream, most software applications will become commodities. For example, a company that wishes to implement a Human Resources application from PeopleSoft will be indifferent to which ASP provides it. The main difference between the offerings of different ASP's will be in their SLA's and their ability to execute their SLA — this is what customers will focus on in choosing their ASP's.

By implementing the present invention, the ASP will have a system that enables it to define its different resources in a single place. The ASP's sales staff will be able to easily tailor an SLA that suits the needs of each customer and charge more for higher level of service, without compromising the ASP's ability to meet the needs of other customers. The ASP may allow the customer to change some of the definitions in the SLA dynamically (for the right price) to accommodate the customer changing needs.

Using the system of the present invention, the ASP manager will be able to identify potential customers that can be offered higher levels of service and additional services.

SUMMARY OF THE INVENTION

According to the present invention there is provided a central management tool, based on a specialized Service Level Agreement Language of Measurement that allows Service Providers (such as ASP's - Application Service Providers) to manage all aspects of the SLA's (Service Level Agreements) signed with its customers, and to track the actual service level delivered. Accordingly,

the ASP can provide timely and reliable reports to its customers on the service level delivered, compared with the service level agreed upon in the SLA. Another feature provided by the present invention, hereinafter called Oblicore, is a calculation of penalties to be credited to the customer in case the targets have not been met.

The system of the present invention is based on a language that enables ASP's and users to define their own ways to compute and monitor the service-level. ASP's using such a tool will be able to define and manage much more complex SLA's with their customers, on a customized level. This tool computes service-level in such a way that it can be used by other programs to measure and control that service level. The invention allows maximum flexibility in describing those service-levels, so that each customer can describe his or her own individual method of measuring service level (or even more than one such method). The present invention provides a software means to calculate the service level of individual customers, in such a way as to provide accurately monitored and controlled customized service.

The heart of the present invention is a service-level language that contains formulas. Each such formula describes how to compute some service-level value from measurements collected by the ASP. These measurements are usually collected from various tools that measure resources the ASP uses to supply service to its customers. Each such formula, written in the language of the present invention, can be loaded into the server computer memory, and from there it may collect measurements from measurement tools, and subsequently calculate the service level. The results of these computations can be analyzed, saved and monitored. Furthermore these results can be used to generate various summaries and reports that are used to ensure the smooth maintenance of customer relations, contracts, resource allocation and system development.

BRIEF DESCRIPTION OF THE DRAWING

The invention is herein described, by way of example only, with reference to the accompanying drawings, wherein:

FIGURE 1 is an illustration of the location of the present invention.

FIGURE 2 is an illustration of the system architecture according to the present invention.

FIGURE 3 is a flow chart representing the 3-tier structure of the present invention.

FIGURE 4 illustrates additional system components in the construction yard.

DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention relates to a system for enabling Service providers (including NSP – Network Service Providers, ISP – Internet Service Providers, etc, and primarily ASP – Application Service Providers), hereinafter referred to ASP's, to manage Service Level Agreements (hereinafter referred to as SLA's) in a highly efficient way. The preferred embodiment of the present invention is discussed in detail below. While specific configurations are discussed, it should be understood that this is done for illustration purposes only. A person skilled in the relevant art will recognize that other components and configurations may be used without departing from the spirit and scope of the invention. The present invention is of a system for applying unique formulas to Application Service providers (ASP) computers, in order to effectively manage Service Level Agreements (SLA). Specifically, the present invention can be used to allow ASP's to compute, monitor and control service-levels that it supplies to its customers, on a personalized basis.

SLA (Service Level Agreement) is a way for a service provider and a customer to set a contract that establishes the obligations of the service provider to the customer.
The purpose of this agreement is to give ASP's a tool to define SLA's with their customers and to monitor the actual service level given to a customer against the SLA.

The system supports very granular definitions of service levels: Different targets may be set for every customer, for every aspect of service level, and for every application at any point in time. Defining Service Level Agreement requires a very flexible system. The present invention's solution has been designed to meet the requirements of different ASP's providing a wide range of applications and services to disparate customers. The defining of service domains is done by the system administrator, who defines the combination of resources that should be measured for each service domain. The administrator can control the frequency of measurements performed by the monitoring tool, the frequency in which the present invention will receive the data from the monitoring tool and the rate of aggregation performed by the present invention when the information is stored in the database.

The system is based on an internal SLA language, named Service Level Agreement Language of Measurement (referred to hereinafter as SLALOM), that supports the various aspects of SLA handling: service domains, aggregation rules, penalties, etc. in a flexible manner. This language contains formulas, wherein each formula describes how to compute some service-level value from measurements collected by the ASP. These measurements are usually collected from various tools that measure resources that the ASP uses to supply service to its customers. Each such formula, written in SLALOM, can be loaded into the server computer memory, and from there it may collect measurements from measurement tools, and subsequently calculate the service level. The results of

these computations can be analyzed, saved and monitored. Furthermore these results can be used to generate various summaries and reports that are used to ensure the smooth maintenance of customer relations, contracts, resource allocation and system development. The system administrator does not have to know this language, yet it can be used with a very intuitive and easy to use user interface.

The principles and operations of such a system according to the present invention may be better understood with reference to the drawing, and the accompanying descriptions, wherein:

Referring to **Figure 1**, it can be seen how the present invention sits in between the Application Service providers and customers, managing the Service Level Agreements. The SLA **11** is the agreement signed between the ASP **10** and the customer **12**. The ASP utilizes the present invention, also referred to as Oblicore **13**, to automatically manage the SLA **11**. The present invention **13** also manages billing **14**, Quality of Service, Customer Relation Management and help-desk functions **15**, and sales **16**.

Referring to **Figure 2**, the system is comprised of the following architectural components:

User Interface 21

The user interface **21** is multi lingual Web based interface which displays only the menus allowed for each end user according to his or her security level and role. The User interface **21** tier is a web-interface. It has been developed using ASP (Active server pages) with VbScripts.

Security Layer 22

This layer is responsible for securing the Oblicore system data by enforcing a security policy restricting users from performing operations they are not permitted to perform, and from viewing information they are not permitted to access.

SLA (Service Level Agreement) Manager 33

This component manages the administrative work of the SLA: It where relevant.

SLA DATABASE 32

This database contains the SLA definitions that target the amount of service level promised to the customer per a certain service domain, application and a certain time slot (or a group of time slots). This database contains the information that the SLA Manager uses.

SLA Engine 31

This component is responsible for processing the data in the SLA DB **32** and generating maps of the promised service level for a customer or a group of customers within a period of time or a time slot.

CSL Engine 28

This component processes the measurements and events reported by the M/O tools **27** after being translated into Oblicore resources id's by the M/O Plug-ins **26**.
The information is calculated, aggregated and then stored in the CSL DB **29** reflecting the measured service level actually provided by the ASP at a certain time interval.

CSL DB 29

The CSL database **29** contains the Calculated Services Level measurements and events calculated and aggregated by the CSL engine **28**. The aggregation method, as well as the aggregation time defined in the SLA DB **32** as a part of the formula of the given rule.

Data Consolidator 30

This component is responsible for processing the information from both the SLA engine **31** and the CSL engine **28** and returning the deviation of the given service from the promised one, and the penalty declared for that deviation.

Reports Generator 23

This component produces reports based on information received from the SLA engine **31** (targets), from the CSL engine **28** (actual service levels) and from the Data consolidator **30** (deviation from the promised service level).

Monitoring/ Operational (M/O) Tools 27

These hardware and software tools, located outside of the Oblicore system, monitor the various ASP's resources: Network, Servers, Help desk, etc. There might be more than one M/O tool **27** since different tools can monitor different resources and can take different measurements on the same resources.

Monitoring/Operational Plug-in 26

This component plugs into a certain M/O tool **27** and translates the measurement/events from this M/O tool **27** into a uniform Oblicore message and forwards this message into the CSL engine **28**. This component is independent of the Oblicore system and therefore contains a layer defining the connection to the Oblicore system (through the CSL engine **28**) on the one hand and the M/O tool connection layer, which defines the method of retrieving the measurements/events from the M/O tool **27** on the other hand). The core of the plug-in translates the M/O measurement/events into Oblicore messages.

The Plug-in can connect either to a Network monitoring tool or to CRM tools and in the future into Monitoring tools for consulting, training, security, etc., to include virtually all the services that may provided by the ASP to its customers.

The way the plug-in interacts with the M/O tool **27** depends on the way the M/O tool **27** functions. Some tools offer an API (application program interface), which can be accessed by the plug-in. Other ways include gathering the data from a log file or a database. If those methods are not sufficient, then information can also be gathered by screen scraping of the M/O tool management screens.

The system may include off-the-shelf plug-ins to the most commonly used M/O tools **27**. An SDK will be provided for implementing plug-ins to home grown applications or rarely used tools.

Infrastructure Manager 24

This component is responsible for holding the information about the map of resources, i.e. what is the role of each resource, where is it connected, and which user/users are influenced by it. This

component is crucial since it allows the system to find the resources that should be monitored for each customer, in order to compute that customer's service level.

Optimization Engine 25

This component runs sophisticated algorithms and supports "what if" scenarios that are aimed at optimizing the allocation of resources by the ASP to better meet the overall commitments of the ASP to its customers. This is in order to increase customer satisfaction and reduce penalties.

The following is an example of a SLA defined between an Application Service Provider and a specific customer:

ASP predefinitions

As a preparation for using the system the system administrator may define general information regarding the ASP which is described below:

1) **Customers.** The system administrator may define in the system a list of customers and their details, or extract this information from the ASP's CRM (Customer Relationship Management) database. The Service Level Agreement's may be attached to the customers later on (it is also possible to define a new Customer while feeding in his or her SLA).

2) **Customer groups.** The system enables the ASP to group Customers together into logical Customer Groups. A single customer may be a member of more than one group. Multi-customer reports may be applied to a Customer Group to reflect summaries of the members' information.

3) **Applications.** The system administrator defines in the system a list of Applications that are provided by the ASP. This list will help attaching SLA's to specific services.

4) **Application groups.** A set of Applications may be defined to form a logical Application Group. The definition of Application Groups such as "Office" (Word, Excel etc.) or "ERP applications" (SAP, Oracle application etc.), may streamline the process of defining SLA's, when instead of listing many Applications, one can refer to an Application Group. Also, summary reports may be applied to an Application Group.

5) **Service domains.** A Service Domain (also referred as a 'Domain') is a specific aspect of the service level agreed upon in the SLA. In general, the ASP's system administrator defines the Service Domains, but Oblicore will supply a set of predefined Service Domains with the system. A Service Domain is defined by the following attributes:

) **Domain name** – is used for identifying the Domain.

i) **Domain description** – is used for describing the Domain semantics.

ii) **Unit** – the measurement unit that represents the semantics of the service level under this Domain (for example: seconds, percent, BPS etc.).

v) **Relation to target** – what would be considered a deviation from the Target – a higher service level than the Target or a lower service level.

) **Aggregation rule** – what function should be applied to a set of service level indications over the Target Period, in order to calculate the measured service level over this time period (into one value to be compared to the Target). For example: average, min, max,

count, sum.

6) **SLA sections.** Each Service Domain is defined under a SLA Section, which is a logical group of domains, sorted by their semantics. The report generator may produce group summaries, in order to sum up the service level in all the Domains in a SLA section. For example: under the "Help desk" SLA Section one may find the "Response time of handling level 1 trouble tickets", "Response time of handling level 2 trouble tickets" and "Success of helpdesk sessions" Service domains. The Domains under a certain SLA Section do not necessarily share the same unit of measurement or semantics.

7) **Infrastructure.** The system administrator should enter the mapping of ASP resources allocated to customers. This Infrastructure database will enable the system to track the actual service level to a specific customer. This information can be retrieved by the system automatically from a monitoring tool, if the monitoring tool contains this information.

8) **Galleries**. Since many SLA's and SLA properties are defined based on a standard (Platinum/ Gold/ Silver/ Bronze) or repeat themselves, Oblicore's system enables the ASP to define Galleries of SLA's and SLA properties, and choose from these Galleries when creating a new SLA. When choosing a template from a Gallery one will be asked to fill in some empty fields – unique details that cannot be predefined. It is always possible to modify the new item if the template doesn't totally fit. The following Galleries are available:

) **SLA's gallery** – a gallery of full SLA's. Provides full templates for SLA's and enable the ASP to create a new standard SLA with the push of a button.

i) **Rules gallery** – a set of templates for SLA Rules (including the Domain Formula but not the Target for example).

ii) **Timeslots gallery** – a set of predefined commonly used timeslots, such as: weekends, holidays etc.

SLA structure

1. **Customer details**. The Customer may be a person or an enterprise. Oblicore will hold the Customer's basic properties in its database.

2. **SLA general details:**

i) **Effectiveness dates**. As far as a SLA is concerned, the date of creation is not necessarily the effective date. The effectiveness dates indicate exactly when the SLA is valid, and correspondingly when it is invalid.

ii) **Storage periods.** Two periods defining the durability of the two data storages of the system:

iii) **Raw data storage** - minimally aggregated service level data, by which the system is able to provide low level drilldowns into the service level provided to the Customer; very expensive with storage resources.

iv) **Aggregated data storage** - maximally aggregated service level data, by which the system is able to provide high level reports (deviations, penalties, averages etc.); These may be kept for long terms, when storage resources are concerned.

3. Locale – specific details as for the locality of the Customer under this SLA. It contains the following parameters:

Language

Currency

Time zone

Daylight saving time

Date format

Number format

4. Applications allocation. The ASP should define which Applications or Application Groups would be provided to a Customer under a certain SLA.

5. Timeslots definition. In order to define different service level objectives per different time slices, an SLA should contain a well defined set of Timeslots, of two types:

-A weekly timeslot – a collection of time ranges, contiguous or non-contiguous, in a resolution of 10 minutes, within a week. No specific dates are defined within such a Timeslot.

- A yearly timeslot – a collection of specific dates, contiguous or non-contiguous, and collection of time ranges (in a resolution of 10 minutes), contiguous or non-contiguous. In general, whenever a specific point of time falls into both a yearly Timeslot and a weekly Timeslot – the yearly Timeslot is in effect.

6. Rules definition. A Rule is a combination of several parameters, defining together a service level objective for a certain service, in a certain time. The Rule also defines the Penalty for deviations from the agreed service level. No two Rules are allowed to exist at the same time, regarding the same

exact service (same Application, same Service Domain). A Rule is defined by the following parameters and attributes:

Rule name – makes it easy to recognize the specific Rule.

Rule description – makes it easy to understand the Rule's semantics.

Service domain – each Rule is attached to one Service Domain. The Domain imposes the semantics of the service objective defined by the Rule, and therefore the unit of measurement and the meaning of deviation from the Target. The method of calculating the provided service level is determined specifically for each Rule by the Domain Formula.

7. Domain formula – a complex formula describing the exact method of calculating the actual service level under the relevant Domain. Different Customers and ASP's may measure service level differently in general, or even differently for various Timeslots and Applications within the same SLA. Therefore the exact method of processing the measurements is defined on a Rule basis. The formula also defines the needed frequency of the various measurements participating in the service level calculation, and the resolution of the calculated service level storage (Raw Data Storage).

8. Related applications – the Applications (or Application Groups) to which this Rule applies. Thanks to this, different service objectives may apply to different Applications. If no Application is related to the Rule – it holds for al the Applications provided to the Customer under the specific SLA.

9. Related timeslots – the Timeslots to which this Rule applies, i.e. – when does this Rule hold. Overlapping Timeslots (of the same form: weekly or yearly) are allowed within the same Rule. If no Timeslot is related to the Rule – it holds permanently by default.

10. Target – This is the promised service level for the applied Timeslots and Applications. The Target is to be tested each Target Period. The unit of the Target is inherited from the Domain definition.

11. Target period – a time interval to which the Target is related as a whole. In other words – for each Rule, once every Target Period, the provided service level is calculated and checked against the Target to calculate deviations from the agreed and/or Penalties.

12. Penalty formula – a compound formula defining the amount of money to be credited to the Customer as a Penalty for deviations from the promised service level once every Target Period. The formula is based on mathematical operations on the Target, the provided service level within the Target Period, and the ratio between the two. Also, the formula may contain cases and conditions to refine the Penalty rate. Remark: if the ASP doesn't use Penalties, this option will be hidden.

<u>SLA Example</u>

In this example, the actual definitions are in the gray boxes. The italic text is just a more readable (human language) explanation of the gray boxes.

The service level agreement is signed between the ASP and Alladin Lamps and assoc. (any company)

The SLA is for the period between 02/10/2001 and 02/09/2002.

Customer: Aladdin Lamps and assoc.

Effective date: 02/10/2001

Expiration date: 02/09/2002

According to the agreement, Raw measurement data will be stored for the period of one month. For periods longer than one month, aggregated measurement data will be stored. The aggregated data will be stored for the period of 15 months.

The data will be presented to the user of this SLA using the format of the USA/CA locale

Raw data storage period: 1 month

Aggregated data storage period: 15 months

Locale: USA/CA

This agreement concerns the following three applications/Application Groups: Office, SAP, Remedy.

Applications: Office, SAP, Remedy.

Timeslots:

Three different timeslots are defined in this agreement.

1. Weekly timeslot: Weekdays: Monday through Friday 8am to 5pm.
2. Weekly timeslot: Weeknights: Monday through Friday 5pm to 8am.
3. Yearly timeslot: New Year: 12/31/2001 through 1/1/2002.

The following rules describe the main points agreed between the ASP and the customer, as to the service level the ASP should provide the customer with:

Rules:

a) **Name**: Weekdays uptime

During Weekdays, the uptime of the system will be no less than 98%. In case the ASP will not satisfy this requirement, it will pay the customer a penalty according

to the penalty formula described later. For calculating the uptime, the system will be considered UP when at least one web server and one application server and two routers are up. All the resources will be sampled each ten seconds. The system will allow drill down information up to the resolution of ten minutes. The aggregation of the data is done by using the average function.

b) **Service domain**: Uptime (aggregation rule: average).

c) **Domain formula**: at least 1 web server, one application server and two routers of the customer are up when measuring once every 10 seconds. Write into the CSL the percentage of the measurements that resulted 'true' out of all the measurements within each 10 minutes.

d) **Related applications**: SAP, Excel, Remedy

e) **Related timeslots**: Weekdays

f) **Target**: 98%

g) **Target period**: 1 day.

h) **Penalty formula**: In case of deviation of 0 to 20 percent: penalty=$0.3*deviation per seat. In case of deviation of more than 20 percent: penalty=$1 per seat

i) **Name**: Weekdays min uptime for Word

j) **Service domain**: Minimal uptime (aggregation rule: min).

k) **Domain formula**: at least 2 web servers, one application server and one router of the customer are up when measuring once every 20 seconds. Write into the CSL the percentage of the measurements that resulted 'true' out of all the measurements within each 10 minutes.

l) **Related applications**: Word

m) **Related timeslots**: Weekdays

n) **Target**: 82%

o) **Target period**: 1 month.

p) **Penalty formula**: penalty=30$ + (100-provided service level)*2$.

Referring to **Figure 3**, the 3-tier-structure system of the present invention is illustrated.

1. User interface tier **40** is a web-interface. It has been developed using ASP (Active server pages) with VbScripts. This is represented by the GUI interface.

2. Business logic tier **50** has been developed using C++, as COM objects. This is represented by the Business Objects.

3. Database tier **60** is a standard relational SQL. An Oracle DB is currently being used. This is represented by the Data Objects **61** and Data Source **62** sections of Figure 3.

As can be seen in **Figure 3**, the system consists of the following main subsystems, which are described below:

- Measurement sub-system **70**
- SLA sub-system **71**
- Reports sub-system **72**
- Policy sub-system **73**
- Logging sub-system **74**
- Secured level **75**

The present invention includes the following modules

There are several modules or sub-systems that comprise the present invention. These modules, as can be seen from **Figure 3**, work in cooperation in order to run the system. It is recommended to look at Figure 3 as well as figure 4 in order to follow the following descriptions. **Figure 4** illustrates additional system components in the construction yard **100.**

Measurement sub-system 70

Main purposes of the sub-system:

- Collect measurements from external information sources.
- Normalize measurements to unified form.
- Calculate and store service level information according to SLF, operated on the received measurements.

Methodology:

The sub-system includes Readers components. Each such reader is responsible of communicating with a specific information source, collecting measurements from the information source, normalizing the measurements to the unified form called messages and sending them to the External Dispatcher **101.**

The External Dispatcher **101** receives messages **102** from the Readers, and sends them to the Internal Dispatchers **103** that registers those messages to the objects that may register for receiving specific type of measurements.

For each service level formula **104**, the Formula Construction Yard **100** creates a Formula object grouped by the formula group **105**. The formula object holds the tally for making a CSL record from the messages that arrived. Each formula object registers itself in his Internal Dispatcher **103** for the relevant messages and the last one registers the messages in the External Dispatcher **101.** The formula object receives the messages from the Internal Dispatcher **103**, calculates the service level according to the formula **104**, and store it in the database.

The External Dispatcher **101** is responsible for controlling and timing the operations of the Formula objects.

Measurement tool 82:

This software monitors the resources on the network, and is capable of querying them for different parameters. There might be more than one measurement tool since different tools can measure different resources or even different measurements on the same resource.

Oblicore 1.5 Release is dedicated to one Measurement tool called XACCT.

Reader 81:

The Measurement Reader **81** gets measurements and events from the Measurement Tool **82,** translates them to a uniform measurement record and sends them forwards to the Dispatcher **80** on the Oblicore Server.

Every Measurement Reader **81** has a layer specifically configured for the Measurement tool **82** that the reader is designed for.

The layer describing the connection between the Reader **81** and the Dispatcher **80** on the Oblicore Server is the same for all the Readers.

The differences between the Readers will be in the Measurement tool Connection layer and in the configuration of the filtering and translation of the data form the Measurement tool **82** to Oblicore.

External dispatcher 101:

The external dispatcher **101** is responsible to receive all the messages **102** from the readers **81**, and forward them to those formula groups **105** that need them. But the dispatcher **80** does more than this. It is also responsible for arranging the arriving messages in the right (chronological) order, so that the formula groups can assume the messages arrive in the same order they were created. Another role of the dispatcher **101** (as can be guessed from its name in the drawing) is to create timer events. As mentioned earlier, the dispatcher **101** forwards the messages **102** it receives to those formula groups **105** that need them. It may be asked, how does the dispatcher **101** know where should he forward the message? Another rather curious question is how does it know what timer events are needed. The answer to both of those questions is that formula groups that want to receive some message or timer event, must register that desire with the dispatcher **101**. Moreover, when a formula group **105** does not want to receive some message any more, it must un-register this message at the dispatcher **101**.

The dispatcher **101** receives messages from many readers. Each reader may run on different machine, and receive its measurements from different measurement tools **82**. This might cause a situation where messages arrive to the dispatcher **101** in a different order than the order they were created. The dispatcher **101** must be able to arrange those messages in the chronological order before forwarding them. This is done by maintaining a priority queue of messages waiting to be forwarded, and delaying each message in that queue for a couple of minutes. This way when a message arrives

one minute later than it should have, the messages that were created after it still wait in the queue, and this message can be sent before them.

Another important role of the dispatcher **101** is to serve as the main timer of the system. Each formula group **105** that wants to receive timer events each specified amount of time, registers that timer event with the dispatcher **101**. In order to create those timer events, the dispatcher **101** does not use a real clock, but rather watches the time stamps on the arriving messages, and uses them to measure the time. This way the timer events arrive synchronized with the messages.

Formula groups 105:

Formula groups **105** are the objects that receive the messages from the dispatcher **101**. Each such group contains an internal dispatcher object **103**, and many formula objects **104**. Each formula group object **104** runs on a single computer in a single process, but there might be many such formula groups, and each can run on a different computer. This makes the system more scalable since the formula group might be CPU demanding.

Internal dispatcher 103:

The internal dispatcher **103** object is very similar to the external dispatcher object **101**. It receives messages from the external dispatcher **101** and forwards them to those formula objects **104** that need it. Each formula object **104** that wants to receive some kind of message, must register that message with the internal dispatcher **103**. The internal dispatcher **103** must then register this message with its external brother (the internal dispatcher registers each message with the external one only once and not for each formula). When a formula unregistered some kind of message with the internal

dispatcher **103**, the dispatcher is responsible to un-register this message with the external dispatcher **101**if no other formula needs it.

Formula 104:

A formula object **104** represents a calculation made on the measurements and saved to CSL **106**. The formula is a tree, constructed from a formula written in SLL language (also known as SLALOM), and which is saved in the SLA database. Each formula object needs many different measurements in order to perform its calculation. When the formula object **104** is constructed, it must register each such measurement with the internal dispatcher **103**, and from that moment each such measurement will be dispatched to the right node in the formula tree.

The construction yard object does the construction of the formula objects from the text of the formula.

CSL database 83:

The CSL database **83** (some call it RSL) contains Calculated Service Level measurements. Each measurement is aggregation of some simple measurements from the measurement tools. The aggregation method, as well as the aggregation time is defined in the formula objects **104**. Each record is either aggregated over a given amount of time (such as 10 minutes), or in some other way (such as from one failure to another).

Each record in the CSL **106** contains the following fields:

- From time – the beginning of the time period this record represents.
- To time – the end of this period.

- Formula ID – the id of the formula that wrote this record
- val1,val2 ,…,val5 – five values that can be calculated by the formula (most formulas will calculate only one of those values.

Backup 107:

The backup object is used to perform backup of the measurements. The backup is done for three main reasons:

- In order to restore the state of the formula objects in case of a crash.
- In order to allow the user a drill down to the level of the raw measurements.
- For debug purposes.

The backup object registers itself with the dispatcher to receive all the messages (including all the timer messages). Then, upon receiving those messages the backup object stores them to the MSL database.

Restoration of each formula state is done as following:

Retrieving all the messages from the MSL **107** that came after the last timer event, and sending them to the formula.

ERASeR 108:

This Module is responsible for deleting old records from the CSL database **83**.

The time limitation on CSL record defined in the SLA by the user how determined for how long to save the detailed measurements.

This module activated automatically every X time and removing all the records connected to formula object that contains the time limitation value.

SLA sub-system 71

Main purposes of the sub-system:

- Allow the manipulation of SLA's, including insertion of new SLA's to the system, viewing and updating existing SLA's.
- Allow the manipulation of all SLA related data, such as customers, applications, domains, time-slots, formulas, etc.
- Supply information about SLA's for reporting components.

Methodology:

An SLA is composed of a set of RULES, where each RULE defines a specific service level requirement.

A RULE is composed of the following elements:

- An aggregation rule that defines the way that single service levels of a number of time slices are aggregated to form a service level over a period of time.

  An aggregation rule is one of the functions **sum**, **avg**, **weight_avg**, **min**, **max** and **count**.

  The aggregation rule is operated on a specific SLF.
- A target for service level.
- The relation between the service level and the target.
- The period of time that the target relates to.
- The TIME-SLOTs that the rule applies to.

For example, to represent the requirement that "The average uptime on weekdays will be above 98%, on a hour basis calculation", the following data is needed:

SLF	= UPTIME
RULE	= avg
TARGET	= 98
RELATION	= >
TIME_SLOT	= weekdays
CALC_PERIOD	= 1 hour

Components:

SlaManager **85**:

Supply methods for manipulation of SLA's, including insertion of new SLA's to the system, viewing and updating existing SLA's.

AdminManager:

Supply methods for manipulation of all SLA related data, such as customers, applications, domains, time-slots, formulas, etc.

SlaProcessor:

Supply information about SLA's for reporting components.

SlaReader **86** will be able to answer questions like:

- "What is the service level promised to customer x, in domain y, during time slot z"

- "What are the service levels promised to customer x, in application y domain z, during period t1-t2"

Reports sub-system 72

Main purposes of the sub-system:

- Produce system reports and graphs

Methodology:

Reports sub-system **72** gathers information about SLA's and CSLs and produces various reports, graphs, summaries and analysis.

Generation of a report/graph is done in 2 phases:

- Gathering information (From SlaProcessor and/or CslProcessor)
- Using an external reporting tool to produce the output.

Components:

ReportsGenerator **23**:

Supply methods for producing all available reports.

DataConsolidator **30**:

Supply methods for retrieving summarized information about deviations and penalties.

Policy sub-system 73

Main purposes of the sub-system:

- To allow setting a security policy that determine what operation may user, or group of users perform.
- To supply online policy validation for secured components.

Methodology:

Manages users, groups and authorizations

Each user can belong to a number of groups (or none)

A user can be given authorizations for specific operations.

A group can be given authorizations for specific operations.

A user is permitted to perform an operation if either he, or one of the groups that he belongs to have the authorization for the operation.

Components:

PolicyValidator **86**:

PolicyManager **87**:

Logging sub-system 74

Main purposes of the sub-system:

- Supply a logging mechanism for all other sub-systems

Methodology:

Components:

Logger:

<u>Secured level sub-system 75</u>

<u>Main purposes of the sub-system:</u>

- Supply secured access for system components

<u>Methodology:</u>

For every component that needs secured access, a parallel secured component should be developed.

The secured components supply the same interface as the parallel unsecured component wraps the component and add security validation before activating any function.

The secured components use the PolicyValidator, of the Policy sub-system to validate operations.

<u>Components:</u>

SecuredSLAProcessor

SecuredSLAAdminManager

SecuredSLAManager **90**

SecuredPolicyManager **91**

SecuredCSLProcessor

SecuredInfrastructureManager

As seen in Figure 2 and 3, the reports generator **23** generates both reports and summaries as follows:

Reports

The system enables the generation of highly detailed and specific reports for various dimensions of system activity. These reports are generated by the Reports Generator **23**, in the Business Objects **50** section of the system. Following is a list of reports to be included in the system:

Single SLA Reports

Agreed service level of a certain domain [and a certain application] over a period of time.

Agreed service level for each timeslot [relevant to a period of time] of a certain domain [and a certain application].

Provided service level of a certain domain [and a certain application] over a period of time.

Provided service level of a certain timeslot in a certain domain [and a certain application] over a period of time.

Agreed vs. provided service level of a certain domain [and a certain application] over a period of time.

Agreed vs. provided service level of a certain timeslot in a certain domain [and a certain application] over a period of time.

Agreed vs. provided service level for each timeslot [relevant to a period of time] of a certain domain [and a certain application].

Deviation from the agreed service level of a certain domain [and a certain application] over a period of time.

Deviation from the agreed service level in a certain timeslot in a certain domain [and a certain application] over a period of time.

Penalty for deviation from the agreed service level of a certain domain [and a certain application] over a period of time.

Penalty for deviation from the agreed service level in a certain timeslot in a certain domain [and a certain application] over a period of time.

Deviation from the agreed service level in each domain in a certain SLA section over a period of time.

Penalty for deviation from the agreed service level in each domain in a certain SLA section over a period of time.

Penalty for deviation from the agreed service level for each SLA section over a period of time.

Domains Reports

Average/minimum/maximum/variance of agreed service level for each domain [for a certain application] over a period of time (for all the customers).

Average/minimum/maximum/variance of deviation from agreed service level for each domain [in a certain application] over a period of time (for all the customers).

Average/minimum/maximum/variance of penalty for deviation from agreed service level for each domain [for a certain application] over a period of time (for all the customers).

Application Reports

Average/minimum/maximum/variance of deviation from agreed service level for each application [in a certain domain] over a period of time (for all the customers).

Average/minimum/maximum/variance of penalty for deviation from agreed service level for each application [in a certain domain] over a period of time (for all the customers).

Customers Reports

Provided service level of a certain domain [and a certain application] for each customer over a period of time.

Deviation from the agreed service level of a certain domain [and a certain application] for each customer over a period of time.

Penalty for deviation from the agreed service level of a certain domain [and a certain application] for each customer over a period of time.

Deviation from the agreed service level of a certain SLA section [and a certain application] for each customer over a period of time.

Penalty for deviation from the agreed service level of a certain SLA section [and a certain application] for each customer over a period of time.

Overall Reports

Average/minimum/maximum/variance of deviation from agreed service level [in a certain application] over a period of time (for all the customers, all domains).

Average/minimum/maximum/variance of penalty for deviation from agreed service level [in a certain application] over a period of time (for all the customers, all domains).

Penalty for deviation from the agreed service level for each of the N most mal-treated customers over a period of time (all the domains, all the applications).

Summaries

Per-SLA Summaries

Average/minimum/maximum/variance of agreed service level in a certain domain [and a certain application] over a period of time.

Average/minimum/maximum/ variance of provided service level in a certain domain [and a certain application] over a period of time.

Average/minimum/maximum/variance of provided service level in each timeslot of a certain domain [and a certain application] over a period of time.

Average/minimum/maximum/variance of deviation from service level of a certain domain [and a certain application] over a period of time.

Average/minimum/maximum/variance of deviation from service level in each timeslot of a certain domain [and a certain application] over a period of time.

Average/minimum/maximum/variance of penalty for deviation from service level of a certain domain [and a certain application] over a period of time.

Average/minimum/maximum/variance of penalty for deviation from service level in each timeslot of a certain domain [and a certain application] over a period of time.

Per-domain Summaries

Updated list of top mal-treated customers in each domain.

Updated list of top well-treated customers in each domain.

Per-application Summaries

Updated list of top mal-treated customers of each application.

Updated list of top well-treated customers of each application.

Overall Summaries

Managers' report: total penalty for all customers (each from it's last payment date), mal-treated customers (penalty or deviation), well treated customers (penalty or deviation), penalty / deviation for each SLA section.

Advantages of the present invention:

Application Hosting and Network Bandwidth:

Quality of Service (QOS) systems enable the utilization of bandwidth according to a predefined set of rules. Load balancing systems enable intelligent utilization of several machines in order to run several applications at once. Systems such as these will benefit from being integrated with the SLA layer. The SLA layer will be able to maintain the correct level of service in real time. Instead of predefined rules, the Optimization engine will be able to monitor the current state of the system and to decide on the preferred configuration that meets the different SLA's in the most optimal way. The most important input for these decisions is the actual fee charged for services delivered at, above, or below the agreed upon service target. For example, during a time of high network contingency it's clear that customers which should receive reduced bandwidth are those for which the ASP will incur the least financial damage.

Marketing and Sales:

When the ASP industry becomes mainstream, most software applications will become commodities. For example, a company that wishes to implement a Human Resources application from PeopleSoft (http://www.peoplesoft.com) will be indifferent to which ASP provides it. The main difference between the offerings of different ASP's will be in their SLA's and their ability to execute their SLA — this is what customers will focus on in choosing their ASP's. By implementing Oblicore, the ASP will have a system that enables it to define its different resources in a single place. The ASP's sales staff will be able to easily tailor an SLA that suits the needs of each customer and charge more for higher level of service, without compromising the ASP's ability to meet the needs of other customers. The ASP may allow the customer to change some of the definitions in the SLA dynamically (for the right price) to accommodate the customer changing needs. Using Oblicore system, the ASP manager will be able to identify potential customers that can be offered higher levels of service and additional services. The ASP may charge more

Technical Support:

Customer Relationship Management (CRM) systems provide the ASP with a way to manage all aspects of its interaction with its customers, particularly in conjunction with help desk and technical support. The SLA layer will provide input to the CRM system, both in terms of customer prioritization and SLA targets. For example, the SLA layer will provide information on how quickly calls should be answered for each customer.

Reporting:

The system provides detailed and up-to-the-minute reports on various aspects of service levels delivered to each customer compared with service level guaranteed including penalties if incurred. The reports can be accessed by the customer using any Web Browser. The level details

presented by the system to the customer is defined by the system administrator. These reports serve as a very important factor in increasing the confidence of the customer in his service provider.

Billing:

Based on the actual service level delivered, Oblicore performs a monthly calculation of penalties to be credited to each customer for all the service aspects agreed to in the SLA. This information is passed to the ASP's billing system.

ASP Management:

Because of Oblicore's tight integration with the different services that the ASP offers, the ASP manager is able to obtain a broad overview of operations at all service levels. This will enable the manager to control the different aspects of the operation and to optimize resources. Oblicore does not focus on building the peripheral systems, but instead are focused on creating a robust infrastructure that support these systems and future systems.

Glossary

The following terms are defined for the purposes of this document:

- Domain – a logic category for grouping rules. All the targets of a domain share the same unit, and therefore may be compared.
- Formula – a compound expression representing the method of calculating a single CSL value out of different messages of different resources.
- Measurement - a value representing a status of a resource.
- Event - a value representing a change of a resource's status.
- Message: an event or measurement, formatted into a unified structure and runs through the system.

- CSL Value – a value representing a category of service level, at a certain time period. It is calculated by a single formula, and is used as the basic unit of an aggregation rule.

- Resource – a measurement's subject – usually a hardware component or an object of external interfaced software.

- Indication – a measurement's predicate – the specific question posed to the resource by the Measurement Tool.

- Infrastructure – the map of ASP's resources, their relations and allocations to customers and applications.

- Customer – a person or an enterprise potentially connected to the ASP by an SLA.

- User – a person who potentially logs into the system through his or her web browser. Users may be the customers' end users, the ASP's staff or others. Each user obtains a login password, by which he/she is granted permissions in the system.

- ASP Domain – a group of users defined by their organizational identification. For example – all the ASP staff is of the same ASP domain, whereas a customer's end users are of another.

- Application – software provided to customers by the ASP. An application may function as a resource, be involved in formulas, or may be considered a logical category for grouping SLA rules.

- SLA Rule – an outline of a commitment to a customer as for the service level it shell obtain under a certain domain, application and timeslot, and the methods of measuring this service level.

- Aggregation rule – a function defining the way of calculating a periodic service level out of a set of CSL values.

- Measurement tool – an interfaced application used for gathering measurements and events.

- SLA – Service Level Agreement – a collection of definitions as for the service level promised to a certain customer.

- Timeslot – a collection of time spans outlining a compound time period, in which a certain service level is required. Timeslots differentiate between SLA rules.

- Time Span – a contiguous period of time.

- Penalty – compensation paid by the ASP to the customer for deviating from service level commitments.

- Compensation – extra payment added by a customer to the periodic pay to the ASP, if a certain target has been exceeded positively (i.e. – a better service than required has been given).

- Target – the quantity of service level promised to a customer at a specific time slot and under a certain service category.

- Target Period – a time interval ("month", "day" etc.) at which the target service level is required. The target period dictates the penalty period (i.e. – how often the measurements should be summarized and compared to a target in order to calculate the periodic deviation and penalty).

- Formula Frequency – an interval indicating how often a CSL value should be calculated and written into the database.

Other embodiments of the present invention include being used in any tool or system that measures service-level, or any other calculation based on measurements performed on resources. It can be used in tools that perform such calculations not only for ASP's, but for ISPs, or for any other service provider that has some automatic tools that measure its resources.

While the invention has been described with respect to a limited number of embodiments, it will be appreciated that many variations, modifications and other applications of the invention may be made.

APPENDIX:

SLALOM - Service Level Agreement Language Of Measurement

Grammar

Each formula is a set of commands. Each command is either an assigment expression, a send event expression, or a save CSL expression. There should be only one save CSL expression in each formula, and it must be the last one.

```
Formula                    ->    (assigment-Expression |
                                  SendEvent-Expression)*
                                  SaveCSL-Expression
```

Assigment expression is an assigment of some expression into a variable. There may be only one assigment expression into each variable in the formula. The assigment expression also defines the variable. Each variable must be defined before it is used, and thus there can be no loops in the definitions.

```
Assigment-Expression       ->    Variable := Expression;
```

Send event expression is a function that receives three parameters. This function creates events every given amount of time. The first parameter is a timer event that tells the function when to create events. The second parameter is an expression that contains the value of the event that will be created, and the third parameter is the name of that new event.

```
SendEvent-Expression       ->    SendEvent( Timer-Event,
                                            Expression,
                                            Event-Name);
```

The SaveCSL expression is a function that saves values into the CSL every given amount of time. It receives six parameters. The first is the timer event, which tells the function when to store the data into the CSL. The other five parameters are expressions that hold the five values that will be saved to the five value fields in the CSL.

```
SaveCSL-Expression         ->    SaveCSL( Timer-Event,
                                          Expression,
                                          Expression,
                                          Expression,
                                          Expression,
                                          Expression);
```

The timer event is usually an event created by the timer every given amount of time. "Usually", because it can also be created from a normal expression, i.e. whenever this expression gets a new value.

```
Timer-Event            ->     GetTimerEvent(integer [,string]) |
                              ConvertToTimerEvent(expression)
```

The event name is any legal token name, prefixed by an ampersand.

```
Event-Name             ->     &Token-Name
```

An expression is any sub–formula that has a value.

```
Expression             ->     (Expression) |
                              Expression Binary-Operator Expression |
                              Function |
                              Aggregator |
                              Constant |
                              Variable |
                              Get-Event-Expression
```

An operator is a function performed on two expressions, which has an infix syntax. There are several operators defined in the language. Their meaning is the same as their default meaning in languages like C++. The precedence of those operators is the same as in C++.

```
Binary-Operator        ->     + | - | * | / |
                              > | < | == | != | >= | <=
```

A function is another way to create an expression out of one or more other expressions. There are several kinds of functions that get different number of parameters.

```
Function               ->     Unary-Function |
                              Binary-Function |
                              Trinary-Function |
                              Multinary-Function

Unary-Function         ->     Unary-Function-Name(Expression)

Binary-Function        ->     Binary-Function-Name(Expression,
                              Expression)

Trinary-Function       ->     Trinary-Function-Name(Expression,
                              Expression, Expression)

Multinary-Function     ->     Multinary-Function-Name(Parameter-List)
```

The following functions are defined:

- NOT(x) – returns true if x is false and false otherwise.
- ABS(x) – returns the absolute value of x.
- POW(x,y) – returns the yth power of x.
- MOD(x,y) – returns the reminder when dividing x by y.
- IF(x,y,z) – returns y if x is true. Otherwise returns z.
- SUM(x1,...,xn) – returns the sum of x1 ... xn.
- COUNT_TRUE(x1,...,xn) – returns the number of its arguments that have a true value.
- AND(x1,...,xn) – returns true when all of its parameters are true.
- OR(x1,...,xn) – returns true when one of its parameters is true.
- AVG(x1,...,xn) – returns the average value of its parameters.
- MIN(x1,...,xn) – returns the minimal value of its parameters.
- MAX(x1,...,xn) – returns the maximal value of its parameters.
- PERCENT_TRUE(x1,...,xn) – returns the percent of its parameters that have a true value.

```
Unary-Function-Name        ->   NOT | ABS
Binary-Function-Name       ->   POW | MOD
Trinary-Function-Name      ->   IF
Multinary-Function-Name    ->   SUM | COUNT_TRUE | AND | OR |
                                AVG | MIN | MAX | PERCENT_TRUE
```

The parameter list is a list of expressions. It can be either statically constructed in the formula text (explicit-parameter-list) or it may be constructed dynamically each time the formula is evaluated (implicit parameter list).

```
Parameter-List             ->   Explicit-Param-List | Implicit-Param-List
```

The explicit parameter list is just a list of expressions separated by commas.

```
Explicit-Param-List        ->   [Expression [,Expression]*]
```

The implicit parameter list is a function that is evaluated into a list of expressions separated by commas by the formula preprocessor.
This function receives three parameters. The first one is a variable name. The second is an expression that contains this variable name in some place where a resource id is expected, and the third one is a list of resources. After preprocessing, this function is evaluated into a list of expressions of the same form as its second parameter, but in each such expression, the variable name (given in the first parameter) is replaced by one of the resource id in the third parameter.

```
Implicit-Param-List    ->    #for_each(Variable, Expression,
                             Resource-List)
```

The resource list is a list of resource ids. It can be either a list of explicit resource ids in parentheses, a function that returns a set of resources, or a set operation between two other resource lists.

```
Resource-List          ->    Empty-Resource-Function |
                             Unary-Resource-Function |
                             Binary-Resource-Function |
                             Resource-Set-Operation |
                             (Resource-Id+)
```

An empty resource function is a function that receives no parameters and returns a set of resource ID's.
There are several such functions defined in the language, but there will probably be others.

```
Empty-Resource-Function ->   #ALL_ROUTERS | #ALL_SERVERS | #ALL_HUBS |
                             #ALL_APLICATIONS | #ALL_RESOURCES | ...TBD
```

A unary resource function is a function that returns a set of resource ID's by one parameter, which can be either a string or an integer.

```
Unary-Resource-Function ->   #Unary-Res-Func-Name(Res-Func-Param)

Res-Func-Param          ->   String | Integer
```

There are many unary resource functions, and many others will be probably added later. Among those that are defined today, the one that is the most powerful is GET_RESOURCE_BY_SQL, which receives some SQL statement that should return a recordset of resource ID's and returns those ID's.

```
Unary-Res-Func-Name     ->   ROUTERS_OF | SERVERS_OF | HUBS_OF |
                             APLICATIONS_OF | RESOURCES_OF |
                             GET_RESOURCES_BY_TYPE |
                             GET_RESOURCES_BY_SQL | ...TBD
```

A binary resource function is a function that returns a set of resource ID's by two parameters, each of which can be either a string or an integer.

```
Binary-Resource-Function->    #Binary-Res-Func-Name(Res-Func-Param,
                              Res-Func-Param)
```

Currently there is only one such function defined, but it seems there would be other such functions later on.

```
Binary-Res-Func-Name    ->    GET_RES_BY_TYPE_AND_OWNER | ...TBD
```

A resource set operation, is a function that receives two sets of resource ID's and returns another such set.

```
Resource-Set-Operation  ->    #Res-Set-Func-Name(Resource-List,
                              Resource-List)
```

There are three set operations defined in the language. UNION returns the union of the two sets. INTERSECTION returns their intersection, and SET_SUBTRUCT returns all resource ID's that are in the first set but not in the second.

```
Res-Set-Func-Name       ->    UNION | INTERSECTION | SET_SUBTRUCT
```

An aggregator is a special kind of function. It receives one parameter, but performs operations with the value of that parameter over time. Each time the value of the parameter is changed, the function performs some calculation over it. All the aggregation functions start with @.

The following aggregators are defined:

- SUM – returns the sum of all the values the parameter got over time.
- COUNT – returns the number of values the parameter got over time.
- COUNT_TRUE – returns the number of time the parameter got a true value.
- AND – returns true if all the values the parameter got were true.
- OR – returns true if one of the values the parameter got was true.
- AVG – returns the average value of the parameter over time.
- MIN – returns the minimum value of the parameter during the time.
- MAX – returns the maximum value of the parameter during the time.
- LAST – returns the last value the parameter got.
- PERCENT_TRUE – returns the percent of the times the value of the parameter was true.
- INTEGRATE – returns the integral of the parameter, i.e. the area under the graph that represents the value of the parameter over time.

```
Aggregator              ->    @Aggregator-Name(Expression)

Aggregator-Name         ->    SUM | COUNT | COUNT_TRUE | AND | OR |
                              AVG | MIN | MAX | LAST | PERCENT_TRUE
```

A constant is a number that can not change during the evaluation of the formula. It is either given as a literal in the body of the formula, or it is a #NUM_OF_ITEMS function which returns the number of items in a resource list, and which is computed before evaluating the formula.

```
Constant                ->    Integer | Float |
                              #NUM_OF_ITEMS(Resource-List)
```

A variable is a name prefixed by a $ sign. It represents some part of a formula that has an intermediate value. Before using a variable it must be defined. The definition of a variable is done by assigning some expression to it.

Variable	->	**$**<u>*Token-Name*</u>

Get event expression is a function that returns a value of the last event that it received. Each time it receives an event, it notifies the function that calls it that it has a new value, and thus that function is recomputed.
There are two kinds of get event functions:

- GetEvent - receives measurements from a measurement tool based on the type of mesurement and the resource id of the measured resource
- GetEventByName - receives events created inside of the formula by SendEvent command. It receives the event that have a given name.

Get-Event-Expression	->	<u>*Get-External-Event*</u> | <u>*Get-Internal-Event*</u>
Get-External-Event	->	**GetEvent(**<u>*Event-Type*</u>**,** <u>*Resource-Id*</u>**)**
Get-Internal-Event	->	**GetEventByName(**<u>*Event-Name*</u>**)**

The following are the basic type definitions used in the above definitions

```
Event-Type          ->    String
Resource-Id         ->    Integer
Token-Name          ->    Alpha(Alpha | Digit | _ )*
Integer             ->    [+|-] Digit+
Float               ->    [+|-]Digit+.Digit*
String              ->    "(^")*"
Alpha               ->    A | B | … | Z |
                          A | b | … | z
Digit               ->    0 | 1 | … | 9
```

Examples

Uptime

1. Weekdays: 99% of the time at least 80% of the client's servers are up and all of the routers. Check the condition each 10 seconds.

```
$routers_percent := PERCENT_TRUE(
                        #FOR_EACH(
                              $x,
                              GetEvent("UT",$x),
                              #ROUTERS_OF("arye")
                        )
                    );

$servers_percent := PERCENT_TRUE(
                        #FOR_EACH(
                              $x,
                              GetEvent("UT",$x),
                              #SERVERS_OF("arye")
                        )
                    );

$routers_ok := ($routers_percent == 100);

$servers_ok := ($servers_percent > 80);

SendEvent(
      GetTimerEvent(10,"sec"),
      &ut_ok,
      AND($routers_ok,$servers_ok)
);

SaveCSL(
      GetTimerEvent(10,"min"),
      @PERCENT_TRUE(
            GetEventByName(&ut_ok)
      ),
      0,
      0,
      0,
      0
);
```

2. Weekends: 50% of the time at least (50% of the routers and 100% of the servers) or (100% of the routers and 50% of the servers) are up; out of the rest of the time, at least 40% of the (routers and servers) but no less then one of each is up.

```
$num_routers_ok := COUNT_TRUE(
                        #FOR_EACH(
                              $x,
                              GET_EVENT("UT",$x),
                              #ROUTERS_OF("arye")
                        )
                   );

$num_routers := #NUM_OF_ITEMS(#ROUTERS_OF("arye"));

$num_servers_ok := COUNT_TRUE(
                        #FOR_EACH(
                              $x,
                              GET_EVENT("UT",$x),
                              #SERVERS_OF("arye")
                        )
                  );

$num_servers := #NUM_OF_ITEMS(#SERVERS_OF("arye"));

$percent_routers_ok := ($num_routers_ok / $num_routers) * 100;

$percent_servers_ok := ($num_servers_ok / $num_servers) * 100;

$side1 := OR(
          AND(
               ($percent_routers_ok == 100),
               ($percent_servers_ok > 50)
          ),
          AND(
               ($percent_routers_ok >= 50),
               ($percent_servers_ok == 100)
          )
     );

SendEvent(
     GetTimerEvent(10,"sec"),
     &ev1,
     $side1
);
```

```
$percent_routers_and_servers_ok :=
(($num_routers_ok + $num_servers_ok) / ($num_routers + $num_servers)) * 100;

$side2 := OR(
          AND(
                ($num_servers_ok >= 1),
                ($num_routers_ok >= 1),
                ($percent_routers_and_servers_ok >= 40)
          ),
          $side1
        );

SendEvent(
      GetTimerEvent(10,"sec"),
      &ev2,
      $side2
);

SaveCSL(
      GetTimerEvent(10,"min"),
      @PERCENT_TRUE(GetEventByName(&ev1)),
      @PERCENT_TRUE(GetEventByName(&ev2)),
      0,
      0,
      0
);
```

Bandwidth

1. Busy hours: 90% of the time the bandwidth will be at least of 100 Mb/Sec but not over 200 Mb/Sec. The bandwidth is measured each minute, and is the sum of the bandwidth of router 1234 and router 4321. The bandwidth of each router is the minimal bandwidth measured on that router during the minute.

```
$x1 := @MIN(GetEvent("bw",1234));
$x2 := @MIN(GetEvent("bw",4321));

$x3 := $x1 + $x2;

$x4 := AND(($x3 > 100),($x3 < 200));

SendEvent(GetTimerEvent(1,"min"),&ev,$x4);

SaveCSL(GetTimerEvent(10,"min"),
       @PERCENT_TRUE(GetEventByName("&ev")),0,0,0,0);
```

Response time

1. 90 percent of the time, response time for all applications of the user will be less then 3 seconds, and in 98 percent of the time it will be less than 5 seconds.

```
$x1 := LAST(#for_each($x,GetEvent("rt",$x),#APLICATIONS_OF("John")));
$total := @COUNT($x1);
$gt3 := @COUNT_TRUE(($x1 > 3));
$gt5 := @COUNT_TRUE(($x1 > 5));

SaveCSL(GetTimerEvent(10,"min"),$total,$gt3,$gt5,0,0);
```

MTBF

1. MTBF for server 1111 will be more than 10 days

```
$x1 := NOT(GetEvent("ut",1111));

$x2 := @INTEGRATE($x1);

$x3 := NOT(@DIFFERENTIATE($x1));

SaveCSL(ConvertToTimerEvent($x3),$x2,0,0,0,0);
```

MTTR

1. MTTR for server 1111 will be less that 1 hour

```
$x1 := GetEvent("ut",1111);

$x2 := @INTEGRATE($x1);

$x3 := @DIFFERENTIATE($x1);

SaveCSL(ConvertToTimerEvent($x3),$x2,0,0,0,0);
```

Representation of the formula in memory

In order to evaluate the formula during the run-time of the system, there must be some way to build a computational model of it in the memory. This chapter describes this model.

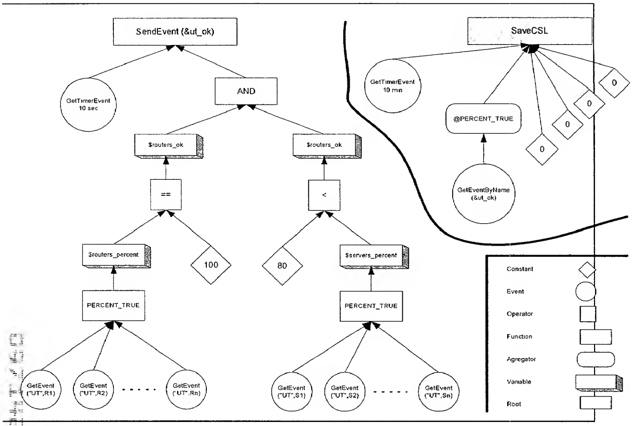
The formula is represented in the memory of the system as a tree of objects. To be more precise, it is a forest. To be yet more precise, each tree in that forest is not a tree but rather a directed graph without directed circles. To make it even worse, the tree can have more than one root.

Each node in this so-called tree represents an expression in the formula language. The children of each node are the sub-expressions of the expression represented by that node. In other words, there is a node for each function, aggregator, constant, variable, operator, get-event, timer-event, send-event, and a save-CSL construct in the language. Each of these nodes, except the variable, send-event and a save-CSL node, has one parent node.

The send-event and a save-CSL nodes have no parent nodes, since they are not contained in another expression. The variable node, on the other hand can have more than one parent (since the variable can be used in more than one expression in the formula).

Following is the first formula example in the previous chapter, and a drawing of the tree representing that formula:

```
$routers_percent := PERCENT_TRUE(
                        #FOR_EACH(
                              $x,
                              GetEvent("UT",$x),
                              #ROUTERS_OF("arye")
                        )
                    );

$servers_percent := PERCENT_TRUE(
                        #FOR_EACH(
                              $x,
                              GetEvent("UT",$x),
                              #SERVERS_OF("arye")
                        )
                    );

$routers_ok := ($routers_percent == 100);
$servers_ok := ($servers_percent > 80);

SendEvent(
      GetTimerEvent(10,"sec"),
      &ut_ok,
      AND($routers_ok,$servers_ok)
);
SaveCSL(
      GetTimerEvent(10,"min"),
      @PERCENT_TRUE(
            GetEventByName(&ut_ok)
      ),0,0,0,0);
```



In this example you can see two trees, one for the send event expression, and one for the save CSL expression. In this example the two are real trees (each node has only one parent) since each variable is used only once, but in the general case, each variable node can have more than one parent.

The circles in this drawing show the flow of events. They point from children towards parents, since the events flow up in this way. The leaves in that tree are either events or constants.

Each time the dispatcher receives a measurement event; it dispatches it to the proper event node. This node receives the value of the event, and tells its parent it has changed. The parent recalculates itself, and notifies its own parent (or parents) that it has a new value, and so on. Aggregators are different in the sense that they do not transform the event upper in the tree.

The SendEvent and SaveCSL are also different, first because they have no parent to forward the event to, and secondly because they perform an operation when they receive a timer event from their first child.

When the SendEvent-node receives a timer event it creates a new event that contains the value stored in its son, and sends that event as a regular event through the dispatcher.

When the SaveCSL-node receives a timer-event, it stores a new record to the CSL database. That new record will contain the five values that are stored in the sons of the SaveCSL-node, the id of the current formula, and a time stamp of the timer-event.

Both, the SaveCSL node and SendEvent node will reset the tree beneath them after processing the timer event. The flow of the reset command is opposite from the flow of the event (from the root towards the leaves).

Each node that receives a reset command, must forward it to its children. Most of the nodes does no more than this. The nodes that really need to do something are the aggregators. Each aggregator has its own way of resetting. The @SUM aggregator for instance resets its counter to zero.

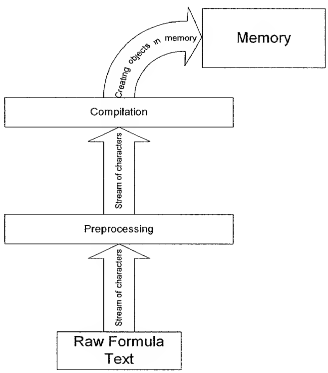
If by now it sounds difficult, wait for the rest. I am just getting warm.

Each node has a value. This value is a floating-point number, and it contains a calculation the node does on its children. This floating-point number can represent a floating point number for functions that accept floating-point numbers (such as the operator +), but it can also represent integers and Booleans. If the function assumes that its parameters are integers (for instance the MOD function), a casting to integer is performed. Casting a floating point number to an integer, means that it is converted to the greatest integer that is not greater than the floating point number. Similarly, when a function assumes to receive a Boolean parameter, the value of the node is casted to a Boolean. Zero converts to false, and any other number converts to true. When a function returns a Boolean, it always should return 1 for true and 0 for false.

When the system starts, each node has a special value called NO_VAL. The node will remain that value until all its children has a value other then NO_VAL. When all the children have a different value, the node calculates itself, and changes its own value from NO_VAL to the calculated value.

Constructing/Destruction the formula in the memory

You have probably noticed that the #for_each functions from the example are not represented in the tree. If you look at the above example, you will notice that the #for_each function in the formula text was replaced by a group of n nodes in the drawing.
In fact, functions that start with the # sign are not real functions. Those functions are replaced by regular function at the first step of parsing the formula. This step is similar to the pre-processing step of the C++ compiler when macros are expanded, so I will refer this step as pre-processing, and to functions that start with a # sign as macros.
After the pre-processing step the formula will contain no macros.
Note that this preprocessing step must be done each time the formula is loaded into the memory, since the macros can be expanded to different result each time (consider for instance the #ALL_ROUTERS macro. If the ASP adds a new router, this macro will return a different result). From this reason it is impossible to perform the preprocessing step in advance, and use its result when loading the formula.
The object that is responsible for loading the formula is the construction-yard. This object does the load in two layers, the first is preprocessing, and the second is compilation.



Preprocessing

The preprocessing is done by a method of the construction-yard, called `preprocess()`.
This function will be written using the flex and bison tools. It will receive a stream of characters, and output another stream of characters, which will be identical to the inputed stream, except that the macros will be replaced expanded.
The outermost macro, is the #for_each macro. All other macros in the formula are contained in the #for_each macro. The preprocessing itself is done in two stages. The first is expanding the resource-list (the third parameter of #for_each), and the second is expanding the #for_each macro itself.
The expansion of the resource-list is done recursively. If it is a list of resources in parenthesis, it is just converted into an STL set of those resources. If it is an XXX-Resource-Function, it runs an SQL statement that should return the resources this resource-function represents, and converts it into an STL set. If the resource-list is a Resource-Set-Operation, it

performs the expansion recursively, on each of its parameters, and performs the specified operation on the two returned STL sets. In any case, the return value of this stage is an STL set of resource-ids.

The second stage in the preprocessing is expanding the #for_each function itself. This is done by looping through all the elements of the set created in the previous stage, and for each resource in that set, writing the text in the second parameter of the function when the variable name is replaced by the value of that resource.

The preprocessor must check for errors, both syntax errors and other errors. The syntax errors should be found in the MMI, and thus are not likely to be fount here, but for rebostness the preprocessor should recognize those errors and handle them. When errors are found, the preprocessor should skip the formula and not crash. Except of syntax errors, the preprocessor must handle other errors. Such errors are errors that result from expanding a resource list to an empty list. Consider a formula that contains the expression #ROUTERS_OF("Zubbi"). It is possible that when the formula is loaded, there are no routers assigned to "Zubbi". If we won't catch this error during the preprocessing step, this will cause a syntax error during compilation.

Compilation

I am not sure that the name "compilation" is the best name for this layer, since it does not create an executable file, like other compilers. It looks more like an interpreter than a compiler, since it converts the formula text to some in-memory model that performs the calculation of the formula. Unfortunately the term "interpretation" doesn't sound good so I chose to call it "compilation".

This stage receives the text of the formulas after all the macros have been expanded, and thus it contains no macros. The building of the tree is done by compiling each statement (assigment, SendEvent-expression, or a SaveCSL-expression), in the order they appear in the formula text.

The exact process of compiling a statement depends on its type, but all of the statement types, involve compilation of expressions.

Expressions are compiled using a recursive function called compile_expression (or any other name we will choose fore it). This function receives the text of an expression, and returns a pointer to the root of a tree representing this formula. The function compiles the formula according to the following rule:

- If it is a constant, a Constant-Node is created for that constant, and a pointer to that node is returned.
- If it is a variable, the compiler looks in a special table that holds all the variable names declared so far, and a pointer to the node of that variable. That pointer is returned.
- If it is a GetEvent function, a GetEvent node is created, and registered with the internal dispatcher. A pointer to that node is returned.
- If it is a GetEventByName function, the name of that function is translated into a unique number using an event table (that table contains names of events, and their translated number). Then a GetEvent node is created and registered for events of type "user defined" and resource-id equal to the number of that event.
- If it is a TimerEvent function, a TimerEvent node is created, and a pointer to that node is returned.
- If it is a compound expression (aggregator, function, operator, or ConvertToTimerEvent function) all of its sub-expressions are compiled recursively, and then a node that corresponds to that expression is created, and all the previously compiled sub-expressions are added to that node as its children. A pointer to that new node is returned.

There should be a special treatment for expressions that contain multiple operators at the same expression level. In this case the operators should be compiled according to their precedence.

Now that we have a function that can compile expressions, the rest of the compilation is simple.

- If we compile an assigment statement, the left side is a variable name, and the right side of the assigment operator is an expression (which we know how to compile). In this case we should create a variable node. Compile the expression on the right side, and link it to the variable as a child. Then we put that variable name and pointer to the variable node in the variables table, and return the pointer to the variable node.
- If we compile a SendEvent statement, we should create a unique number for that event name, and store it in the events table. Then we should create a SendEvent node that sends that event, and then we compile all the expressions that SendEvent uses and link them as children to the SendEvent node.

- If we compile a SaveCSL statement, we do it pretty much the same way as the SendEvent statement. We create a node for the SaveCSL, and then we compile all the six expressions (SaveCSL function receives six parameters), and link those expressions to the SaveCSL node.

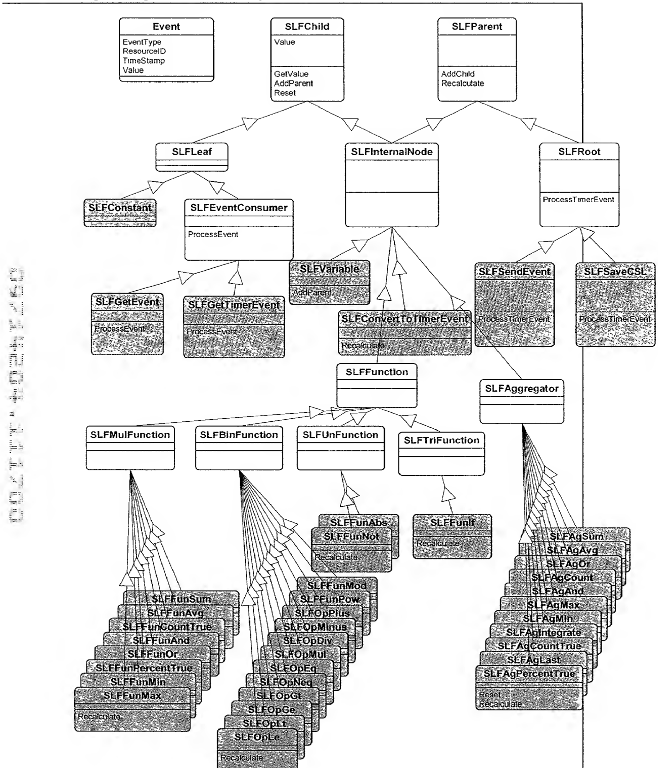
When creating the SaveCSL and SendEvent nodes (which are roots of the trees) we should register those nodes if the formula map. That map is used to access all the roots of trees for a specific formula. We need it in order to be able to destroy a given formula.
In general, there should be no errors in the syntax of the formulas, since the MMI should check it, but the compiler still has to report any errors found in the formula. Formulas that contain errors should not be loaded, but the system should not crash, and continue to load the rest of the formulas.

Destruction of the formula

Destruction of a given formula is done by retrieving all the roots of the trees in the formula, and recursively destroying them. When destroying a GetEvent, GetEventByName, and GetTimerEvent nodes, we must also unregister those nodes with the internal dispatcher. When destructing a variable node, we should remove the parent we came from, from the list of parents of that node. The node should be deleted only when it has no more parents. This will allow us to delete only parts of the formula without affecting the behavior of other parts (trees).

The class hierarchy

The following drawing shows the hierarchy of the classes that construct the formula tree.



The gray shapes are actual classes that form the formula tree. The white classes are abstract classes that the former derive from.

The following table describes those classes in more detail:

Class	Derives from	Class description: Method	Description
SLFParent		This class is a base class for all tree nodes that have children.	
		AddChild	This is a pure virtual function that must be implemented in the descendants of this class. This function is used to add a child node to this node. It receives one parameter of type SLFChild*
		Recalculate	This is also a pure virtual function This function is called by children of that node in order to signal a change that occurred in the child, and that requires the recalculation of the parent.
SLFChild		This class is a base class for all child nodes that can be children of other nodes in the system. This class contains one data member called mValue.	
		GetValue	This function is virtual but not pure virtual. It returns the value of the mValue data member. This function is used by the parent of that node to retrieve its value
		AddParent	This is a virtual function (not pure). It adds a parent to that node.
		Reset	This is a pure virtual function used by the parent of this node to reset all the sub-tree under that node. This function should reset the node itself (only aggregators should really do something to reset themselves), and then it should call reset on each of its children.
SLFLeaf	SLFChild	This class is a base class for all nodes that should be leaves at the formula tree. It has no new methods. It is just a logical class, and may be defined using typedef instead of defining a new class.	
SLFInternalNode	SLFChild SLFParent	This class is a base class for all nodes that are neither leafs nor roots. It has no new methods, and it is used to combine the methods from its two base classes	
SLFRoot	SLFParent	This class is a base class for the two possible root nodes. The SLFSaveCSL and SLFSendEvent.	
		ProcessTimerEvent	This method is used by the GetTimerEvent and ConvertToTimerEvent nodes to notify this node that a timer event occurred.
SLFEventConsumer	SLFLeaf	This node is a base node for nodes that can receive events.	
		ProcessEvent	This event is used by the dispatcher to notice the node about an event it receives. It has one parameter of type pointer to event.
SLFAggregator	SLFInternalNode	This node is a base node for all aggregator nodes.	
		AddChild	This function is an implementation of the pure virtual function defined in SLFParent. It adds a child to the node. If the node already has a child it throws an exception.
SLFFunction	SLFInternalNode	This class is a base class to all function and operator classes. It has no new methods, and its use is just to make the hierarchy more organized.	
SLFUnFunction	SLFFunction	This node is a base node for al the unary functions in the formula. Unary functions are function nodes that have one child node.	
		AddChild	Implementation of the pure virtual method defined in SLFParent
		Reset	Implementation of the pure virtual method defined in SLFParent
SLFBinFunction	SLFFunction	This node is a base node for al the binary functions in the formula. Binary functions are function nodes that have two child nodes.	
		AddChild	Implementation of the pure virtual method defined in SLFParent
		Reset	Implementation of the pure virtual method defined in SLFParent
SLFTriFunction	SLFFunction	This node is a base node for al the trinary functions in the formula. Trinary functions are function nodes that have three child nodes.	
		AddChild	Implementation of the pure virtual method defined in SLFParent
		Reset	Implementation of the pure virtual method defined in SLFParent
SLFMulFunction	SLFFunction	This node is a base node for al the multinary functions in the formula. Multinary functions are function nodes that have many (one or more) child nodes	

		Reset	Implementation of the pure virtual method defined in SLFParent
SLFMulFunction	SLFFunction	This node is a base node for al the multinary functions in the formula Multinary functions are function nodes that have many (one or more) child nodes.	
		AddChild	Implementation of the pure virtual method defined in SLFParent
		Reset	Implementation of the pure virtual method defined in SLFParent
SLFConstant	SLFLeaf		This node represents a constant. It receives a value during construction, and from that moment returns this value when the GetValue method is called.
SLFVariable	SLFInternalNode	This node represents a variable. Unlike other child nodes, this node can have more than one parent. The value of this node is the value of the node beneath it.	
		AddParent	Overloading of the virtual function defined in SLFChild. Adds a parent to a node.
		Recalculate	Implementation of the pure virtual method defined in SLFParent It should get the value of the child and put it in mValue property
		AddChild	Implementation of the pure virtual method defined in SLFParent. Adds a child to that node If a child already exists it throws an exception.
		Reset	Declared in SLFChild. Performs reset on its child.
SLFConvertToTimer Event	SLFInternalNode	This node has one child node and one parent node. Whenever the child calls |Recalculate on this node, this node calls ProccessTimerEvent on its parent. Its parent must be of type SLFRoot.	
		Recalculate	Calls ProcessTimerEvent of the parent.
		AddChild	Implementation of the pure virtual method defined in SLFParent. Adds a child to that node. If a child already exists it throws an exception
		Reset	Declared in SLFChild. Performs reset on its child.
SLFGetTimerEvent	SLFEventConsumer	This class is a node that receives timer events. When it is constructed the timer event is registered for that node.	
		ProcessEvent	Declared in SLFEventConsumer. This method calls ProcessTimerEvent on its parent
SLFGetEvent	SLFEventConsumer	This class represents a node that receives regular events.	
		ProcessEvent	Declared in SLFEventConsumer. When this function is called, it sets the mValue of the node to the value of the event, and calls Recalculate on its parent
SLFSendEvent	SLFRoot	This node has two children. This node represent a SendEvent construct of the language.	
		ProcessTimerEvent	Defined in SLFRoot. This function sends event with the value of its second child After the save it calls reset on its children.
		AddChild	Defined in SLFParent. Adds children to the node. Only two children can be added.
SLFSaveCSL	SLFRoot	This node has six children. This node represent s SaveCSL construct of the language.	
		ProcessTimerEvent	Defined in SLFRoot This function saves a node to the CSL with values of the five last children. After the save it calls reset on its children.
		AddChild	Defined in SLFParent. Adds children to the node. Only six children can be added.
SLFFunIf	SLFTriFunction	Represents the **IF** function.	
		Recalculate	Sets the value of the node to the value of its second child if the value of its first child is true. Otherwise it returns the value of its third child. Then calls recalculate on the parent.
SLFFunAbs	SLFUnFunction	Represents the **ABS** function. Returns the absolute value of its parameter.	
		Recalculate	Sets the value of the node to the absolute value of its child node. Then calls recalculate on the parent.
SLFFunNot	SLFUnFunction	Represents the **NOT** function.	
		Recalculate	Sets the value of the node to the NOT of the value of its child node. Then calls recalculate on the parent.
SLFFunPow	SLFBinFunction	Represents the **POW** function.	

SLFOpPlus	SLFBinFunction	Represents the + operator	
		Recalculate	Sets the value of the node to the X + Y, where X is the value of the first child node, and Y is the value of the second Then calls recalculate on the parent
SLFOpMinus	SLFBinFunction	Represents the - operator	
		Recalculate	Sets the value of the node to the X - Y, where X is the value of the first child node, and Y is the value of the second. Then calls recalculate on the parent.
SLFOpDiv	SLFBinFunction	Represents the / operator.	
		Recalculate	Sets the value of the node to the X / Y, where X is the value of the first child node, and Y is the value of the second. Then calls recalculate on the parent.
SLFOpMul	SLFBinFunction	Represents the * operator.	
		Recalculate	Sets the value of the node to the X * Y, where X is the value of the first child node. and Y is the value of the second. Then calls recalculate on the parent.
SLFOpEq	SLFBinFunction	Represents the == operator.	
		Recalculate	Sets the value of the node to true when X == Y, and false otherwise, where X is the value of the first child node, and Y is the value of the second Then calls recalculate on the parent.
SLFOpNeq	SLFBinFunction	Represents the != operator.	
		Recalculate	Sets the value of the node to true when X != Y, and false otherwise, where X is the value of the first child node, and Y is the value of the second. Then calls recalculate on the parent.
SLFOpGt	SLFBinFunction	Represents the > operator.	
		Recalculate	Sets the value of the node to true when X > Y, and false otherwise, where X is the value of the first child node, and Y is the value of the second. Then calls recalculate on the parent.
SLFOpGe	SLFBinFunction	Represents the >= operator	
		Recalculate	Sets the value of the node to true when X >= Y, and false otherwise, where X is the value of the first child node, and Y is the value of the second Then calls recalculate on the parent.
SLFOpLt	SLFBinFunction	Represents the < operator.	
		Recalculate	Sets the value of the node to true when X < Y, and false otherwise, where X is the value of the first child node, and Y is the value of the second. Then calls recalculate on the parent.
SLFOpLe	SLFBinFunction	Represents the <= operator.	
		Recalculate	Sets the value of the node to true when X <= Y, and false otherwise, where X is the value of the first child node, and Y is the value of the second. Then calls recalculate on the parent
SLFFunAvg	SLFMulFunction	Represents the **AVG** function.	
		Recalculate	Sets the value of the node to the average of the values of its children. Then calls recalculate on the parent.
SLFFunSum	SLFMulFunction	Represents the **SUM** function.	
		Recalculate	Sets the value of the node to the sum of the values of its children. Then calls recalculate on the parent.
SLFFunCountTrue	SLFMulFunction	Represents the **COUNT_TRUE** function.	
		Recalculate	Sets the value of the node to the number of true values of its children. Then calls recalculate on the parent.
SLFFunPercentTrue	SLFMulFunction	Represents the **PERCENT_TRUE** function.	
		Recalculate	Sets the value of the node to the percent of true values of its children. Then calls recalculate on the parent.
SLFFunOr	SLFMulFunction	Represents the **OR** function.	
		Recalculate	Sets the value of the node to true when one of the values of its children is true. False otherwise. Then calls recalculate on the parent.
SLFFunAnd	SLFMulFunction	Represents the **AND** function.	
		Recalculate	Sets the value of the node to true when all of the values of its children is true. False otherwise. Then calls recalculate on the parent.
SLFFunMin	SLFMulFunction	Represents the **MIN** function	

		Recalculate	Sets the value of the node to true when one of the values of its children is true. False otherwise Then calls recalculate on the parent.
SLFFunAnd	SLFMulFunction	Represents the **AND** function	
		Recalculate	Sets the value of the node to true when all of the values of its children is true False otherwise Then calls recalculate on the parent
SLFFunMin	SLFMulFunction	Represents the **MIN** function.	
		Recalculate	Sets the value of the node to the minimum of the values of its children Then calls recalculate on the parent.
SLFFunMax	SLFMulFunction	Represents the **MAX** function.	
		Recalculate	Sets the value of the node to the maximum of the values of its children Then calls recalculate on the parent.
SLFAgSum	SLFAggregator	Represents the **@SUM** aggregator.	
		Recalculate	Adds the value of its child to the member mValue
		Reset	Sets mValue to 0
SLFAgAvg	SLFAggregator	Represents the **@AVG** aggregator	
		Recalculate	Adds one to the value of mCount. Adds the value of the child to mTotal. Sets mValue to mTotal/mCount
		Reset	Sets mTotal to 0 and mCount to 0.
SLFAgOr	SLFAggregator	Represents the **@OR** aggregator.	
		Recalculate	Sets mValue to mValue OR child.GetValue()
		Reset	Sets mValue to false
SLFAgAnd	SLFAggregator	Represents the **@AND** aggregator.	
		Recalculate	Sets mValue to mValue AND child.GetValue()
		Reset	Sets mValue to true
SLFAgCount	SLFAggregator	Represents the **@COUNT** aggregator	
		Recalculate	Adds one to mValue
		Reset	Sets mValue to 0.
SLFAgCountTrue	SLFAggregator	Represents the **@COUNT_TRUE** aggregator.	
		Recalculate	Adds one to mCount if the value of the child is true
		Reset	Sets mValue to 0.
SLFAgPercentTrue	SLFAggregator	Represents the **@PERCENT_TRUE** aggregator.	
		Recalculate	If the value of the child is true, adds one to mTrue. Adds one to mTotal Sets mValue to mTrue/mTotal*100
		Reset	Sets mTotal=mTrue=0
SLFAgMin	SLFAggregator	Represents the **@MIN** aggregator.	
		Recalculate	Sets mValue to the value of the child if it is less than mValue
		Reset	Sets the mValue to NO_VALUE
SLFAgMax	SLFAggregator	Represents the **@MAX** aggregator.	
		Recalculate	Sets mValue to the value of the child if it is greater than mValue
		Reset	Sets the mValue to NO_VALUE
SLFAgIntegrate	SLFAggregator	Represents the **@INTEGRATE** aggregator.	
		Recalculate	Sets the value of mValue to mValue+child.GetVal()*(cur_time-last_time); Sets last_time to cur_time
		Reset	Sets mValue to 0
SLFAgLast	SLFAggregator	Represents the **@LAST** aggregator	
		Recalculate	Sets the value of mValue to the value of the child
		Reset	Does nothing

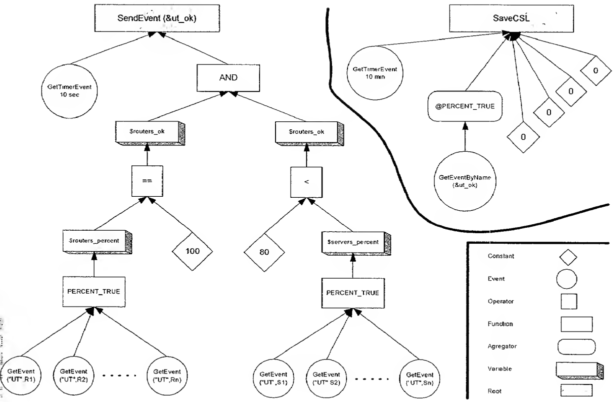
There are other classes that will be used during the construction of the formulas. Such classes are map_of_variables, which holds variable names and pointers to their nodes, and map_of_events. Those classes will not be discussed further here.

Appendix – Formula examples with memory representation

Uptime

3. Weekdays: 99% of the time at least 80% of the client's servers are up and all of the routers. Check the condition each 10 seconds.

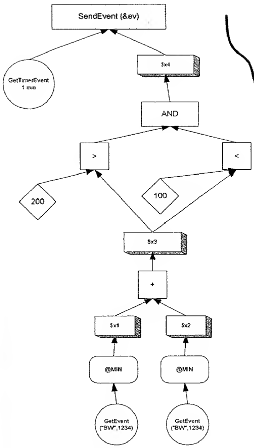
```
$routers_percent := PERCENT_TRUE(
                        #FOR_EACH(
                              $x,
                              GetEvent("UT",$x),
                              #ROUTERS_OF("arye")
                        )
                    );

$servers_percent := PERCENT_TRUE(
                        #FOR_EACH(
                              $x,
                              GetEvent("UT",$x),
                              #SERVERS_OF("arye")
                        )
                    );

$routers_ok := ($routers_percent == 100);

$servers_ok := ($servers_percent > 80);

SendEvent(
      GetTimerEvent(10,"sec"),
      &ut_ok,
      AND($routers_ok,$servers_ok)
);

SaveCSL(
      GetTimerEvent(10,"min"),
      @PERCENT_TRUE(
            GetEventByName(&ut_ok)
      ),
      0,
      0,
      0,
      0
);
```

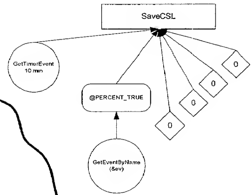

SendEvent (&ut_ok)
GetTimerEvent 10 sec
AND
$routers_ok
$routers_ok
==
<
$routers_percent
100
80
$servers_percent
PERCENT_TRUE
PERCENT_TRUE
GetEvent ("UT",R1)
GetEvent ("UT",R2)
GetEvent ("UT",Rn)
GetEvent ("UT",S1)
GetEvent ("UT",S2)
GetEvent ("UT",Sn)
SaveCSL
GetTimerEvent 10 min
@PERCENT_TRUE
GetEventByName (&ut_ok)
0
0
0
0
Constant
Event
Operator
Function
Agregator
Variable
Root

Bandwidth

2. Busy hours: 90% of the time the bandwidth will be at least of 100 Mb/Sec but not over 200 Mb/Sec. The bandwidth is measured each minute, and is the sum of the bandwidth of router 1234 and router 4321. The bandwidth of each router is the minimal bandwidth measured on that router during the minute.

```
$x1 := @MIN(GetEvent("bw",1234));
$x2 := @MIN(GetEvent("bw",4321));

$x3 := $x1 + $x2;

$x4 := AND(($x3 > 100), ($x3 < 200));

SendEvent(GetTimerEvent(1,"min"),&ev,$x4);

SaveCSL(GetTimerEvent(10,"min"),
    @PERCENT_TRUE(GetEventByName("&ev")),0,0,0,0);
```

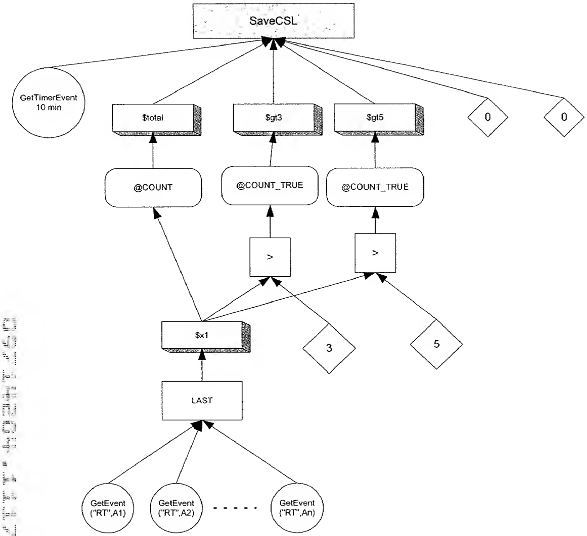




Response time

2. 90 percent of the time, response time for all applications of the user will be less then 3 seconds, and in 98 percent of the time it will be less than 5 seconds.

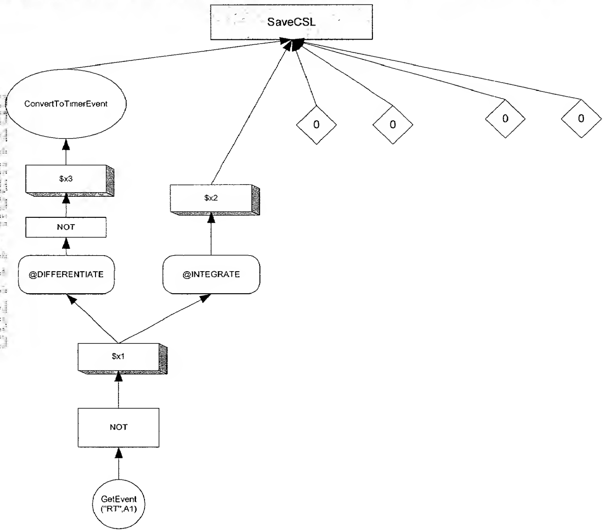
```
$x1 := LAST(#for_each($x,GetEvent("rt",$x),#APLICATIONS_OF("John")));
$total := @COUNT($x1);
$gt3 := @COUNT_TRUE(($x1 > 3));
$gt5 := @COUNT_TRUE(($x1 > 5));

SaveCSL(GetTimerEvent(10,"min"),$total,$gt3,$gt5,0,0);
```


SaveCSL
GetTimerEvent 10 min
$total
$gt3
$gt5
0
0
@COUNT
@COUNT_TRUE
@COUNT_TRUE
>
>
$x1
3
5
LAST
GetEvent ("RT",A1)
GetEvent ("RT",A2)
GetEvent ("RT",An)

MTBF

2. MTBF for server 1111 will be more than 10 days

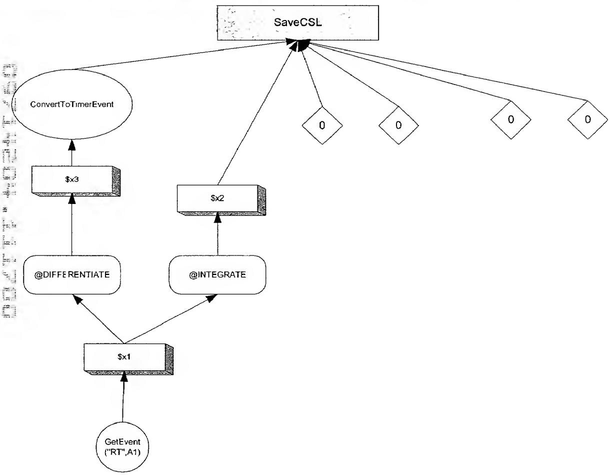
```
$x1 := NOT(GetEvent("ut",1111));
$x2 := @INTEGRATE($x1);
$x3 := NOT(@DIFFERENTIATE($x1));
SaveCSL(ConvertToTimerEvent($x3),$x2,0,0,0,0);
```



MTTR

2. MTTR for server 1111 will be less that 1 hour

```
$x1 := GetEvent("ut",1111);
$x2 := @INTEGRATE($x1);
$x3 := @DIFFERENTIATE($x1);
SaveCSL(ConvertToTimerEvent($x3),$x2,0,0,0,0);
```



WHAT IS CLAIMED IS:

1. A system for automatically managing Service Level Agreements comprising:

i. A database module for storing data related to the Service Level Agreements;;

ii. An engine module for processing said data from said database module; and

iii. A management module for managing said database module and said engine module.

2. The system of claim 1, further comprising a security layer for securing data generated by said database module, said engine module and said management module.

3. The system of claim 1, further comprising a reports generator for generating SLA reports based on said engine module.

4. The system of claim 1, wherein said management module includes:

A. A SLA Manager for creating and updating Service Level Agreements; and

B. An Infrastructure Manager for finding resources to be monitored for each customer with said Service Level Agreement.

5. The system of claim 1, wherein said engine module includes:

C. A SLA engine for generating maps of promised service level for a customer;

D. A CSL engine for processing measurements and events of said service level, as reported by Monitoring/ Operational tools; and

E. An Optimization Engine for supporting "what if" scenarios for optimizing the allocation of resources for said customer by an Application Service Provider.

6. The system of claim 1, wherein said database module includes:

F. A SLA database containing SLA definitions that target an amount of service level promised to a customer per a certain service domain, application and a certain time slot; and

G. A CSL database that contains Calculated measurements and events of said Services Level.

7. A tool for defining, monitoring and executing Service Level Agreements with customers, comprising:

i. A SLA Manager for creating and updating Service Level Agreements;

ii. A SLA database for containing definitions of said Service Level Agreements, that target an amount of service level promised to a customer; and

iii. A SLA engine for processing data in said SLA database.

8. The tool of claim 7, further comprising:

iv. At least one Monitoring/Operational Tool for monitoring Application Service Providers resources; and

v. A Monitoring/Operational Plug-in for translating measurement and events from said Monitoring/Operational tool into a uniform message and forwarding said message to a CSL engine.

9. The tool of claim 7, further comprising:

vi. A CSL engine for processing measurements and events reported by said Monitoring/Operational tools; and

vii. A CSL database that contains Calculated Services Level measurements and events calculated and aggregated by said CSL engine.

10.The tool of claim 7, further comprising:

viii. A Data Consolidator for processing information from said SLA engine and said CSL engine, and returning a deviation of given service from promised service, and a penalty declared for that deviation;

ix. A Reports Generator for producing reports based on information received from said SLA engine, said CSL engine and said Data consolidator; and

x. An Infrastructure Manager for enabling the tool for finding said Application Service provider resources; and

xi. An Optimization Engine for optimizing allocation of resources by an Application Service provider.

11.A method for enabling at least one Application Service Provider to manage a Service Level Agreement, comprising the following steps:

i. setting up at least one production computer for executing data processing jobs;

ii. setting up at least one computer console for extracting job performance data from said production computer;

iii. setting up at least one production server, connected to said computer console, for storing said job performance data;

iv. setting up at least one maintenance workstation for loading data pertaining to SLA's on said production server;

v. setting up at least one client workstation for automating SLA monitoring and displaying actual performance of said data processing jobs, said SLA performance of jobs, problems, and impacts to downstream jobs to a user; and

vi. setting up a local area network (LAN) for connecting said maintenance workstation and said client workstation to said production server.

12.The method of claim 11, wherein the managing of SLA's further comprises the step of setting up a Service Level Agreement Language of Measurement to operate on said production computer, said computer console, said production server, said maintenance workstation, said client workstation and said LAN.

13.The method of claim 12, wherein said setting up of Service Level Agreement Language of Measurement further comprises:

A. ascribing at least one formulas for describing how to compute some service-level value from measurements collected by the ASP

B. building a computational model of said formula; and

C. constructing of said formula in memory.

14.The method of claim 13, further comprising the step of destructing said formula in memory.

15.A method for defining, monitoring and controlling a Service Level Agreement by means of a language, comprising the steps of:

i. Defining grammar of a formula;

ii. Defining the Semantics of said formula; and

iii. Defining a hierarchy of classes of objects used to build a memory model that computes said formula.

16. The method of claim 15, wherein said memory model is built from said formula text, comprising the steps of:

a. Parsing said formula text; and

b. building a tree of objects representing said memory model of said formula.

17. The method of claim 16, wherein said memory model evaluates said formula during runtime of the system.

ABSTRACT OF THE DISCLOSURE

The present invention describes a system for automatically monitoring and managing Service Level Agreements on behalf of Service providers (such as Application Service providers). The system is based on a specialized SLA language that can translate complex or simple Service Level Agreements into measurable and controllable criterion. The system enables Application Service providers to set up customized Service Level Agreements with customers, and monitor, modify and control all aspects of these agreements, including billing, sales, Customer Relation Management, customer support and Quality of Service. The technology on which the present invention is based is a formula driven language that translates Service Level Agreement details into commands. As such these details can be tracked and processed to produce detailed reports and summaries.

DRAWINGS

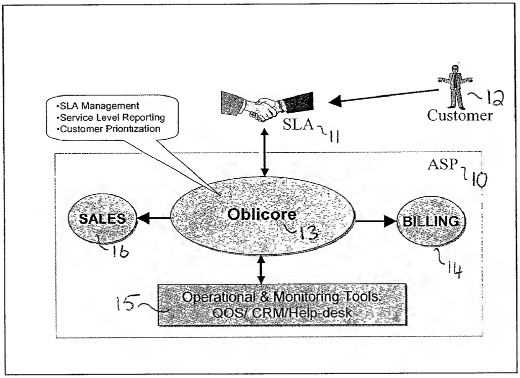


Figure 1

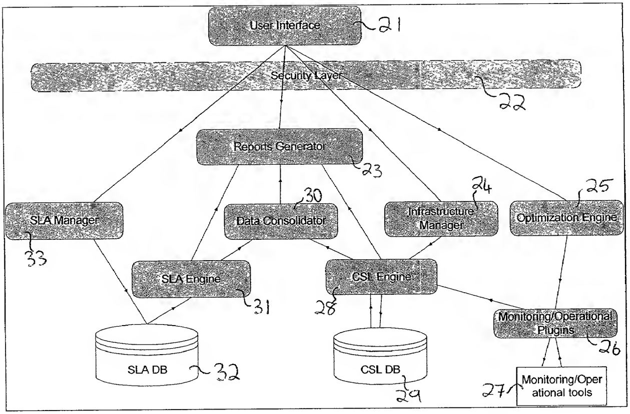

User Interface
21
Security Layer
22
Reports Generator
23
SLA Manager
33
Data Consolidator
30
Infrastructure Manager
24
Optimization Engine
25
SLA Engine
31
CSL Engine
28
Monitoring/Operational Plugins
26
SLA DB
32
CSL DB
29
Monitoring/Operational tools
27

Figure 2

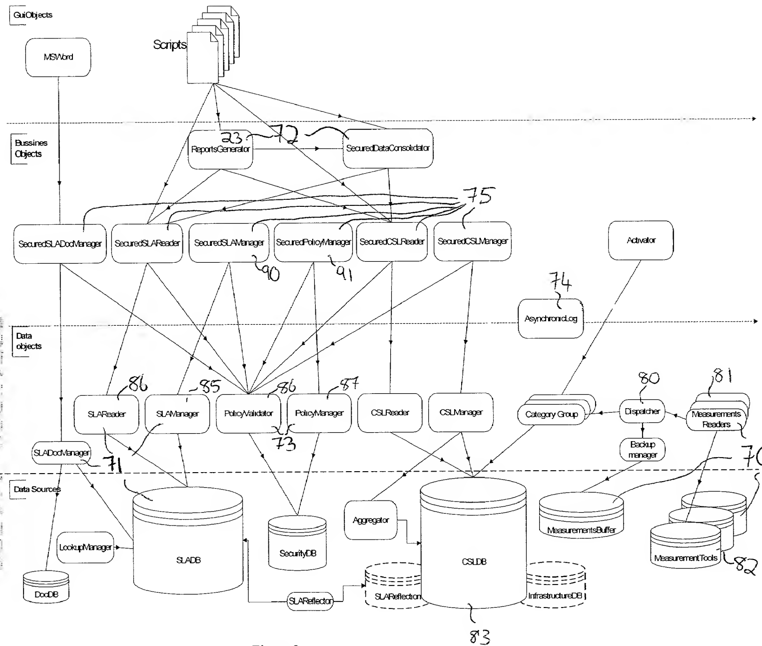
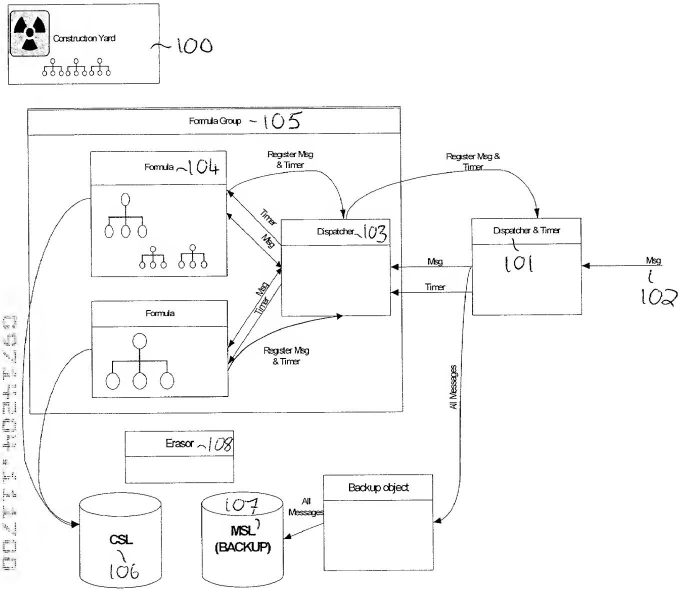

GuiObjects
MSWord
Scripts
Bussines Objects
ReportsGenerator
SecuredDataConsolidator
23
72
75
SecuredSLADocManager
SecuredSLAReader
SecuredSLAManager
SecuredPolicyManager
SecuredCSLReader
SecuredCSLManager
Activator
90
91
74
AsynchronicLog
Data objects
86
85
86
87
80
81
SLAReader
SLAManager
PolicyValidator
PolicyManager
CSLReader
CSLManager
Category Group
Dispatcher
Measurements Readers
Backup manager
73
SLADocManager
71
70
Data Sources
SLADB
SecurityDB
Aggregator
CSLDB
MeasurementsBuffer
MeasurementTools
82
LookupManager
DocDB
SLAReflector
SLAReflection
InfrastructureDB
83

Figure 3



Construction Yard
100
Formula Group
105
Formula
104
Register Msg
& Timer
Register Msg &
Timer
Timer
Msg
Dispatcher
103
Dispatcher & Timer
101
Msg
Timer
Msg
102
Formula
Msg
Timer
Register Msg
& Timer
All Messages
Erasor
108
Backup object
All
Messages
107
MSL
(BACKUP)
CSL
106

Figure 4

Combined Declaration For Patent Application and Power of Attorney

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled **A system and method for analyzing and coordinating Service-Level-Agreements (SLA) for Application-Service-Providers (ASP).**, the specification of which

(check one) ☒ is attached hereto.

☐ was filed on as Application Serial No. and was amended on . I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of this application in accordance with Title 37, Code of Federal Regulations, § 1.56(a).

I hereby claim foreign priority benefits under Title 35, United States Code, § 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)			Priority Claimed	
(number)	(Country)	(Day, Month, Year Filed)	☐ Yes	☐ No
(number)	(Country)	(Day, Month, Year Filed)	☐ Yes	☐ No
(number)	(Country)	(Day, Month, Year Filed)	☐ Yes	☐ No

I hereby claim the benefit under Title 35, United States Code, § 120 of any United States Application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States code, § 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, § 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

(Application Serial No.)	(Filing Date)	Status (patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	Status (patented, pending, abandoned)

I hereby appoint the following attorneys, with full power of substitution, association, and revocation, to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith.

Mark M. Friedman Registration No. 33,883

Address all Correspondence to:

DR. MARK FRIEDMAN LTD.
c/o ANTHONY CASTORINA
2001 JEFFERSON DAVIS HIGHWAY
SUITE 207
ARLINGTON, VIRGINIA 22202

Direct all telephone calls & faxes to:
ANTHONY CASTORINA
Phone (703) 415-1581
Fax (703) 415-4864

Continuation of Combined Declaration For Patent Application and Power of Attorney

I hereby further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statement may jeopardize the validity of the application of any patent issued thereon.

*FULL NAME OF SOLE OR FIRST INVENTOR	INVENTOR'S SIGNATURE	DATE
Arye Barkan		14-November-2000
RESIDENCE	CITIZENSHIP	
79/16 Levanon Street, Ramat Aviv 69345, Israel	ISRAELI	
POST OFFICE ADDRESS		
79/16 Levanon Street, Ramat Aviv 69345, Israel		

*FULL NAME OF SECOND INVENTOR	INVENTOR'S SIGNATURE	DATE
Aviv Freidin		14-November-2000
RESIDENCE	CITIZENSHIP	
22 Bilu Street, Rishon-Lezion 75318, Israel	ISRAELI	
POST OFFICE ADDRESS		
22 Bilu Street, Rishon-Lezion 75318, Israel		

*FULL NAME OF THIRD INVENTOR	INVENTOR'S SIGNATURE	DATE
Lior Musman		14-November-2000
RESIDENCE	CITIZENSHIP	
17/17 Bavli Street, Tel-Aviv 62331, Israel	ISRAELI	
POST OFFICE ADDRESS		
17/17 Bavli Street, Tel-Aviv 62331, Israel		

*FULL NAME OF FOURTH INVENTOR	INVENTOR'S SIGNATURE	DATE
Noam Rotem		14-November-2000
RESIDENCE	CITIZENSHIP	
10/8 Emeq-Ayalon Street, Modiin 71700, Israel	ISRAELI	
POST OFFICE ADDRESS		
10/8 Emeq-Ayalon Street, Modiin 71700, Israel		

*FULL NAME OF FIFTH INVENTOR	INVENTOR'S SIGNATURE	DATE
Dror Treves		14-November-2000
RESIDENCE	CITIZENSHIP	
32 Monash Street, Tel-Aviv, Israel	ISRAELI	
POST OFFICE ADDRESS		
32 Monash Street, Tel-Aviv, Israel		

*FULL NAME OF SIXTH INVENTOR	INVENTOR'S SIGNATURE	DATE
Gal Baram		14-November-2000
RESIDENCE	CITIZENSHIP	
23/1 Hapudim Street, Ramat-Gan, Israel	ISRAELI	
POST OFFICE ADDRESS		
23/1 Hapudim Street, Ramat-Gan, Israel		

*FULL NAME OF SEVENTH INVENTOR	INVENTOR'S SIGNATURE	DATE
Sharon Wagner		14-November-2000
RESIDENCE	CITIZENSHIP	
11 Kibutz Galuyot Street, Netanya, Israel	ISRAELI	
POST OFFICE ADDRESS		
11 Kibutz Galuyot Street, Netanya, Israel		

United States Patent & Trademark Office

Office of Initial Patent Examination -- Scanning Division

SCANNED, #

Application deficiencies were found during scanning:

☐ Page(s) 1 of Drawings were not present for scanning.
(Document title)

☐ Page(s) ____ of ____ were not present for scanning.
(Document title)

☐ Scanned copy is best available.